J.C. FANTOU — G. RIVAUD

30 ROUTINES ASSEMBLEUR

Pour langages évolués et DOS

Avec en plus, 25 procédures d'intégration

Ecran de 43 lignes obtenu grâce à la routine EGA 43 (p.154)



ÉDITIONS RADIO

J.-C. FANTOU — G. RIVAUD

30 ROUTINES ASSEMBLEUR

Pour langages évolués et DOS

Avec en plus, 25 procédures d'intégration

30 ROUTINES pour

Clavier/écran	1.	Vider la mémoire-tampon du clavier
	2.	Obtenir le code ASCII d'une touche
	3.	Activer la touche Caps Lock
	4.	Désactiver la touche Caps Lock
	5.	Activer la touche Num Lock
	6.	Désactiver la touche Num Lock
	7.	Lire l'état des touches spéciales
	8.	Positionner le curseur
	9.	Afficher 43 lignes de texte
Horloge	10.	Chronométrer au 1/18e de seconde
	11.	Lire l'heure
	12.	Lire la date
Haut-parleur	13.	Jouer une note de musique
Carte EGA	14.	Passer en mode graphique EGA
	15.	Délimiter une fenêtre graphique
	16.	Tirer un trait en XOR
	17.	Déplacer un réticule
	18.	Colorier un rectangle en XOR
	19.	Afficher une icône
Fichiers	20.	Protéger un fichier contre l'effacement
	21.	Déverrouiller un fichier protégé
	22.	Cacher un fichier
	23.	Rendre visible un fichier caché
	24.	Sauvegarder une image EGA
	25.	Charger une image EGA
Système	26.	Relancer le système
Souris	27.	Programmer la souris
Port série	28.	Paramétrer le port série
Tablette	29.	Piloter une tablette graphique
Imprimante	30.	Recopier un écran EGA

avec les LANGAGES

- .Basica IBM
- .GWBasic Microsoft
- .Basic compilé (compilateur Bascom)
- .QuickBasic Microsoft
- .TurboBasic Borland
- .Pascal Microsoft
- .TurboPascal Borland
- .Fortran77 Microsoft
- .C & Quick C Microsoft
- .Turbo C Borland

...sans oublier le DOS !_

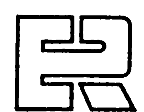
Ecran de 43 lignes obtenu grâce à la routine EGA 43 (p.154)



ÉDITIONS RADIO

J.C. FANTOU – G. RIVAUD

30 ROUTINES ASSEMBLEUR



ÉDITIONS RADIO
189, RUE SAINT-JACQUES - 75005 PARIS
TEL. (1) 43.29.63.70

Éditions Radio - Siège social : 103, boulevard Saint-Michel - 75005 Paris

Imprimé en France par Berger-Levrault, Nancy

Dépôt légal : décembre 1988
Éditeur nº 1124 - Imprimeur : 774987
I.S.B.N. 2 7091 1042 3

Avertissement au lecteur

Ne vous y méprenez pas !... Le présent document ne constitue pas un cours sur l'Assembleur et ne peut en aucun cas se substituer aux nombreux ouvrages écrits sur le sujet.

Il est la suite logique de l'Assembleur 8088/86 et 80286 de H. Lilen et vous propose, à vous qui avez appris à manipuler ce langage, des routines toutes prêtes à être intégrées dans votre langage de programmation favori, qu'il s'agisse du :

- *Basica*
- *GW Basic*
- *Basic compilé*
- *Quick Basic*
- *Turbo Basic*
- *Pascal*
- *Turbo Pascal*
- *Fortran*
- *C Microsoft*
- *Quick C*
- *Turbo C*

Vous découvrirez que, loin d'être normalisés, les langages précités diffèrent les uns des autres, tout du moins du point de vue de la procédure d'appel des routines, par une multitude de petits détails qui, si l'on n'y prend pas garde, constituent autant d'embûches sur le sentier du programmeur néophyte.

Quant aux routines qui vous sont proposées, sachez qu'elles constituent la toile de fond de bien des programmes professionnels, tant en ce qui concerne le graphisme à très haute résolution, que le pilotage des périphériques d'entrée-sortie (souris, tablette, imprimante, etc...).

Comment interfacer des routines assembleur avec les langages évolués

Les langages usuels de développement étant incomplets, voire dépourvus pour certains des instructions que vous jugez indispensables, il importe que vous sachiez les enrichir en leur intégrant, sous la forme de routines écrites en Assembleur, les éléments de programmation qui leur manquent.

Langages évolués

Au cours de cette première partie, l'interfaçage des routines Assembleur sera successivement étudié à partir des langages évolués suivants :

Quelques conseils...

Intégrer des routines écrites en Assembleur à un langage de développement sous-entend que l'on sache déjà les écrire. C'est pourquoi le présent document ne vise nullement à se substituer aux divers ouvrages d'apprentissage écrits sur le sujet. Tout au plus, amène-t-il quelques compléments.

Nombreuses sont les routines qui nécessitent que leur soient transmis des paramètres, ne serait-ce que pour préciser leur action. Ainsi, une routine de dessin de traits ne saurait fonctionner sans indication des coordonnées de début et de fin du trait, la couleur qui doit être la sienne, voire s'il convient de le dessiner en plein ou en pointillé, en mode normal ou complémentaire, etc., etc.

Or, le mode de passage des paramètres varie d'un langage à l'autre. Selon le cas, les paramètres sont expédiés à la routine sous la forme d'adresses ou de valeurs numériques, la pile les recevant dans l'ordre de leur écriture ou encore dans l'ordre inverse, le RET de fin de routine (c'est l'équivalent d'un RETURN en Assembleur...) se devant de tenir compte ou non du nombre et de la nature des paramètres transmis, quand il ne faut pas purement et simplement le supprimer, etc., etc.

Fort heureusement, la situation se simplifie un peu en matière de récupération des valeurs réexpédiées par une routine. Il n'existe (tout du moins dans les langages analysés...) que deux variantes possibles. Selon le cas, les valeurs sont réexpédiées aux adresses figurant dans la pile ou directement affectées lorsqu'il s'agit de fonctions.

Ceci étant, il reste à aborder le point délicat des procédures d'appel des routines. Or celles-ci sont aussi diversifiées dans le détail qu'il existe de langages différents. Cela va du CALL au CALL ABSOLUTE des Basic, en passant par les procédures INLINE des Turbo langages pour terminer par la déclaration de fonctions des langages Pascals et Cs.

Une dernière mise en garde avant d'aborder l'étude détaillée des procédures d'intégration : N'allez surtout pas croire que celles-ci demeurent immuables d'une version à l'autre d'un même langage de développement. Celles-ci peuvent fort bien être totalement différentes, ce qui remet en cause le savoir faire en ce domaine.

BASICA et GW BASIC

Code en tableau

Le Basic interprété (Basica et GW Basic) se prête mal à l'intégration de routines en Assembleur. Néanmoins il existe un certain nombre de méthodes qui ont fait leurs preuves, notamment celle qui consiste à placer les codes-opération de la routine dans un tableau à l'aide d'instructions DATA.

Routine en Assembleur

Pour l'exemple qui va suivre, nous avons choisi une routine d'addition de 2 nombres entiers qui, bien qu'inutile puisque le Basic possède tout ce qu'il faut en ce domaine, va nous permettre de nous assurer que la transmission des paramètres à la routine, via la pile, s'effectue correctement.

```
;           ************************************
;           Routine d'addition ADDITION.ASM
;           ************************************

CODE        SEGMENT BYTE        ; Déclaration du segment de code
            ASSUME  CS:CODE     ; Affectation du segment de code
ADDITION    PROC    FAR         ; Déclaration de procédure FAR
                                ;
            PUSH    BP          ; Sauvegarde pointeur de base
            MOV     BP,SP       ; Transfert pointeur de pile en BP
            PUSH    SI          ; Sauvegarde index source
                                ;
            MOV     SI,(BP+0AH) ; Chargement adresse variable A en SI
            MOV     AX,[SI]     ; Transfert valeur variable A en AX
            MOV     SI,(BP+08H) ; Chargement adresse variable B en SI
            MOV     BX,[SI]     ; Transfert valeur variable B en BX
            ADD     AX,BX       ; Addition de A avec B (résultat dans AX)
            MOV     SI,(BP+06H) ; Chargement adresse variable C en SI
            MOV     [SI],AX     ; Transfert contenu de AX à la variable C
                                ;
            POP     SI          ; Rétablissement index source
            POP     BP          ; Rétablissement pointeur de base
            RET     6           ; ... et retour à l'envoyeur
                                ;
ADDITION    ENDP                ; Fin de la procédure d'addition
CODE        ENDS                ; Fin du segment de code
            END                 ; Fin de la routine
```

Après déclaration et affectation du segment de code (c'est lui qui renferme le code objet de la routine...), la routine d'addition est déclarée en procédure de type FAR.

Ceci étant, les lignes suivantes sauvegardent l'index source SI (on eut pu tout aussi bien employer l'index destinataire DI...) et le pointeur de base BP avant que ne soit attribué à ce dernier la valeur du pointeur de pile SP. Or, c'est cette valeur qui, mise dans BP, va nous servir à gérer les paramètres dans la pile.

Vous noterez que l'on ne transmet pas à la routine les valeurs numériques des paramètres, mais bien au contraire l'adresse où l'on est susceptible de les trouver et que ceux-ci s'empilent dans l'ordre inverse de leur écriture, le dernier paramètre transmis étant affecté du décalage minimum de 06. Ceci explique pourquoi le transfert des paramètres dans les registres AX et BX se passe en deux temps, l'index source recevant l'adresse dans un premier temps, adresse qui sert ensuite à transférer la valeur des paramètres d'entrée dans les registres AX et BX.

Dans ces conditions, le pas de décalage employé au sein de la routine pour lire la pile est de 2 octets, que les paramètres lui soient transmis (et donc les variables employées...) sous la forme d'entiers sur deux octets ou de type simple ou double précision. Ne tombez pas dans le piège qui consiste à croire que le pas de décalage dépend du type de variable choisie : il n'en est rien !...

L'addition étant effectuée et le résultat expédié vers la variable C, il ne reste plus qu'à rétablir dans leur valeur d'origine l'index source et le pointeur de base avant de rendre la main au programme d'appel. Notez au passage que le RET final est suivi du nombre 6, ce qui équivaut à 3 paramètres de 2 octets chacun et assure la gestion correcte de la pile.

Procédure d'intégration au Basic

La routine étant écrite avec un éditeur de lignes, assemblez-la à l'aide du macro-Assembleur MASM de Microsoft en tapant :

```
>MASM ADDITION;
```

Procédez ensuite à l'édition de liens :

```
>LINK ADDITION;
```

Vous obtenez ainsi un fichier ADDITION.EXE qui, bien que n'étant pas exécutable (sinon, c'est le plantage garanti...) correspond à une étape indispensable à l'intégration de cette routine au sein du Basic interprété.

L'ultime étape (tout du moins en ce qui concerne le traitement à effectuer sur la routine...) consiste à transformer ce fichier exécutable en un fichier binaire. Pour ce faire, il vous faut employer l'utilitaire EXE2BIN livré avec le MSDOS qui, comme son nom l'indique (en anglais, s'entend...), assure la conversion d'un fichier EXE en un fichier BIN. Pour ce faire, tapez :

```
>EXE2BIN ADDITION
```

Un rapide coup d'œil au répertoire vous permet de constater que vous venez de créer un fichier ADDITION.BIN de 26 octets de long, c'est-à-dire un fichier ne comportant que les seuls codes-opération de la routine.

Cette préparation de la routine étant (presque...) terminée, venons-en maintenant à la méthode d'intégration dite du code en tableau. Elle consiste, lorsque la routine est courte, à écrire les codes-opération à l'intérieur même du programme Basic sous la forme de DATAs que l'on charge ensuite dans un tableau. Ce faisant, il ne reste plus qu'à déterminer l'emplacement du début du tableau dans le segment du programme à l'aide d'un VARPTR et à exécuter la routine au travers d'un CALL.

La seule difficulté (il faut bien qù'il y en ait une...) consiste à déterminer les valeurs hexadécimales des codes-opération de la routine. Or cela s'obtient en désassemblant le début du fichier ADDITION.BIN à l'aide d'un utilitaire tel que DEBUG livré avec le MS DOS de la machine. On aboutit alors à quelque chose qui devrait ressembler à ceci :

```
C:\GWBASIC>DEBUG ADDITION.BIN
-U
1100:0100 55            PUSH    BP
1100:0101 8BEC          MOV     BP,SP
1100:0103 56            PUSH    SI
1100:0104 8B760A        MOV     SI,[BP+0A]
1100:0107 8B04          MOV     AX,[SI]
1100:0109 8B7608        MOV     SI,[BP+08]
1100:010C 8B1C          MOV     BX,[SI]
1100:010E 03C3          ADD     AX,BX
1100:0110 8B7606        MOV     SI,[BP+06]
1100:0113 8904          MOV     [SI],AX
1100:0115 5E            POP     SI
1100:0116 5D            POP     BP
1100:0117 CA0600        RETF    0006
-Q
```

Comme vous pouvez en juger par vous-même, les codes-opération, au nombre de 26, apparaissent en regard du programme source en Assembleur. Il ne vous reste plus qu'à noter leurs valeurs hexadécimales de façon à pouvoir les transcrire sous la forme de DATAs dans le programme Basic que voici :

Programme d'essai en Basic

Si l'on s'intéresse au fonctionnement de ce programme, on remarque que toutes les variables sont de type entier (ligne 10). Ce faisant, la ligne 20 dimensionne un tableau de 26 octets (13 indices, 2 octets par indice) destiné à recevoir les 26 codes-opération de la routine.

```
10 DEFINT A-Z
20 DIM TABLEAU(12)
30 A=7:B=5:C=0:I=0:CODE=0
40 DECALAGE=VARPTR(TABLEAU(0))
50 FOR I=DECALAGE TO DECALAGE+25
60 READ CODE
70 POKE I,CODE
80 NEXT
90 ADDITION=VARPTR(TABLEAU(0))
100 CALL ADDITION(A,B,C)
110 PRINT"7+5=";C
120 END
130 DATA &H55,&H8B,&HEC,&H56,&H8B,&H76,&H0A,&H8B
140 DATA &H04,&H8B,&H76,&H08,&H8B,&H1C,&H03,&HC3
150 DATA &H8B,&H76,&H06,&H89,&H04,&H5E,&H5D,&HCA
160 DATA &H06,&H00
```

Après avoir déclaré toutes les variables à la ligne 30, on recherche à la ligne suivante l'emplacement du début du tableau que l'on enregistre sous la forme d'un DECALAGE au sein du segment du Basic. Notez que la déclaration préalable des variables est indispensable, sans quoi la localisation du tableau est faussée...

Fort de ce DECALAGE, les lignes 50 à 80 assurent le chargement octet par octet dans le tableau des 26 codes-opération de la routine. Il eut été également possible de charger le tableau en procédant indice par indice, mais cela eut nécessité de prendre les codes-opération deux par deux, de les convertir en décimal pour ensuite les transformer en un entier signé sur 2 octets.

Le tableau étant chargé, il ne reste plus qu'à appeler la routine d'addition au travers d'un CALL ADDITION(A,B,C), en précisant entre parenthèses le nombre de paramètres à lui transmettre. Pour mémoire, il est rappelé que la variable C correspond au résultat de l'addition de A et de B. Vous noterez que la variable ADDITION possède la même valeur de décalage que la variable DECALAGE et qu'il eut été tout aussi possible (mais moins évident...) d'appeler la routine d'addition à l'aide d'un CALL DECALAGE(A,B,C).

Ce programme étant écrit, sauvegardez-le (précaution indispensable si jamais l'ordinateur se plante...) et exécutez-le. Vous devriez aboutir, si tout ce passe bien, à l'affichage de l'addition.

Récapitulatif de la procédure d'appel

Basic interprété (Basica ou GW Basic) Code en tableau		
Instruction d'appel	:	CALL
Paramètres		variables
Transmis sous forme		d'adresses
Empilage dans l'ordre		inverse d'écriture
Procédure de type		FAR
Décalage initial		06H
Pas de décalage		2 octets
Fin de routine	:	RET n(*)
Directives à spécifier	:	aucune

(*) n = Nombre de paramètres transmis × pas de décalage.

BASICA ET GW BASIC

Fichier binaire

La méthode d'intégration précédente n'étant valable que pour des routines courtes (allez donc recopier sans erreur un millier d'octets...), en voici une seconde qui consiste à transformer la routine en un fichier binaire que l'on charge dans un tableau par un classique BLOAD.

Routine en Assembleur

Afin d'illustrer l'exemple qui va suivre, nous avons choisi une routine d'addition de 2 nombres entiers qui, bien qu'inutile puisque le Basic possède ce qu'il faut en la matière, va nous permettre de nous assurer que le passage des informations à la routine, via la pile, s'effectue correctement.

```
;          ******************************
;          Routine d'addition ADDITION.ASM
;          ******************************

CODE       SEGMENT BYTE          ; Déclaration du segment de code
           ASSUME  CS:CODE       ; Affectation du segment de code
ADDITION   PROC    FAR           ; Déclaration de procédure FAR
                                 ;
           PUSH    BP            ; Sauvegarde pointeur de base
           MOV     BP,SP         ; Transfert pointeur de pile en BP
           PUSH    SI            ; Sauvegarde index source
                                 ;
           MOV     SI,(BP+ØAH)   ; Chargement adresse variable A en SI
           MOV     AX,[SI]       ; Transfert valeur variable A en AX
           MOV     SI,(BP+Ø8H)   ; Chargement adresse variable B en SI
           MOV     BX,[SI]       ; Transfert valeur variable B en BX
           ADD     AX,BX         ; Addition de A avec B (résultat dans AX)
           MOV     SI,(BP+Ø6H)   ; Chargement adresse variable C en SI
           MOV     [SI],AX       ; Transfert contenu de AX à la variable C
                                 ;
           POP     SI            ; Rétablissement index source
           POP     BP            ; Rétablissement pointeur de base
           RET     6             ; ... et retour à l'envoyeur
                                 ;
ADDITION   ENDP                  ; Fin de la procédure d'addition
CODE       ENDS                  ; Fin du segment de code
           END                   ; Fin de la routine
```

Après déclaration et affectation du segment de code (c'est lui qui renferme le code objet de la routine...), la routine d'addition est déclarée en procédure de type FAR.

Ceci étant, les lignes suivantes sauvegardent l'index source SI (on eut pu tout aussi bien employer l'index destinataire DI...) et le pointeur de base BP avant que ne soit attribué à ce dernier la valeur du pointeur de pile SP. Or, c'est cette valeur qui, mise dans BP, va nous servir à gérer les paramètres dans la pile.

Vous noterez que l'on ne transmet pas à la routine les valeurs numériques des paramètres, mais bien au contraire, l'adresse où l'on est susceptible de les trouver et que ceux-ci s'empilent dans l'ordre inverse de leur écriture, le dernier paramètre transmis étant affecté du décalage minimum de 06. Ceci explique pourquoi le transfert des paramètres dans les registres AX et BX se passe en deux temps, l'index source recevant l'adresse dans un premier temps, adresse qui sert ensuite à transférer la valeur des paramètres d'entrée dans les registres AX et BX.

Dans ces conditions, le pas de décalage employé au sein de la routine pour lire la pile est de 2 octets, que les paramètres lui soient transmis (et donc les variables employées...) sous la forme d'entiers sur deux octets ou de type simple ou double précision. Ne tombez pas dans le piège qui consiste à croire que le pas de décalage dépend du type de variable choisie : il n'en est rien !...

L'addition étant effectuée et le résultat expédié vers la variable C, il ne reste plus qu'à rétablir dans leur valeur d'origine l'index source et le pointeur de base avant de rendre la main au programme d'appel. Notez au passage que le RET final est suivi du nombre 6, ce qui équivaut à 3 paramètres de 2 octets chacun et assure la gestion correcte de la pile.

Procédure d'intégration au Basic

La routine étant écrite avec un éditeur de lignes, assemblez-la à l'aide du macro-Assembleur MASM de Microsoft en tapant :

```
>MASM ADDITION;
```

Procédez ensuite à l'édition de liens :

```
>LINK ADDITION;
```

Vous obtenez ainsi un fichier ADDITION.EXE qui, bien que n'étant pas exécutable (sinon, c'est le plantage garanti...) correspond à une étape indispensable à l'intégration de cette routine au sein du Basic interprété.

L'avant-dernière étape (tout du moins en ce qui concerne le traitement à effectuer sur la routine...) consiste à transformer ce fichier exécutable en un fichier binaire. Pour ce faire, il vous faut employer l'utilitaire EXE2BIN livré avec le MS DOS qui, comme son nom l'indique (en anglais, s'entend...), assure la convertion d'un fichier EXE en un fichier BIN. Pour ce faire, tapez :

```
>EXE2BIN ADDITION
```

Un rapide coup d'œil au répertoire vous permet de constater que vous venez de créer un fichier ADDITION.BIN de 26 octets de long, c'est-à-dire un fichier ne comportant que les seuls codes-opération de la routine.

Venons-en maintenant à cette seconde méthode d'intégration qui consiste à charger la routine directement dans un tableau à l'aide d'un BLOAD, comme s'il s'agissait d'un fichier binaire ordinaire. Or cela n'est pas possible avec la structure actuelle, car il lui manque l'en-tête indispensable qui lui permet d'être reconnu, et donc d'être chargé, par l'instruction BLOAD du Basic.

Cet en-tête comportant un ensemble de 7 octets représentant respectivement le code du fichier (FD pour un fichier binaire), l'adresse sur 2 octets du segment où il a été prélevé, le décalage (également sur 2 octets), dans le segment et, pour finir, la longueur (toujours sur 2 octets) du fichier binaire sans l'en-tête, c'est donc vers la méthode d'insertion de cet en-tête au début du fichier binaire que va porter notre effort.

Pour ce faire, il vous faut faire appel à l'utilitaire DEBUG qui vous permet de travailler directement en langage machine sur un fichier. Lancez DEBUG avec le fichier ADDITION.BIN et effectuez une première lecture de la mémoire à partir du décalage 100. Vous devriez obtenir quelque chose ressemblant à ceci :

```
>DEBUG ADDITION.BIN
-D100
10D9:0100  55 8B EC 56 8B 76 0A 8B-04 8B 76 08 8B 1C 03 C3
10D9:0110  8B 76 06 89 04 5E 5D CA-06 00 .. .. .. .. .. ..
```

Vous avez reconnu, débutant au décalage 100 et se terminant en 119, la suite des 26 codes-opération de la routine. Cela étant, il vous faut maintenant décaler de 7 octets cette suite de codes de façon à dégager suffisamment de place pour l'en-tête. Après décalage du début de la routine en 107 (ce qui libère 7 octets...), vous devriez aboutir à ceci :

```
-M100 119 107
-D100
10D9:0100  55 8B EC 56 8B 76 0A 55-8B EC 56 8B 76 0A 8B 04
10D9:0110  8B 76 08 8B 1C 03 C3 8B-76 06 89 04 5E 5D CA 06
10D9:0120  00 .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
```

Il vous faut maintenant introduire l'en-tête de 7 octets à compter du décalage 100. Pour ce faire, point n'est besoin de vous préoccuper des adresses du segment et du décalage devant figurer dans cet en-tête, l'emploi d'un DECALAGE à la suite d'une instruction BLOAD dans le programme Basic ci-après prenant le pas sur ces valeurs.

Seuls comptent le code du fichier binaire (FD) et la longueur sans en-tête (26 octets, soit 1A en hexadecimal...) du fichier binaire. Pour le reste, n'importe quelle valeur fera l'affaire. De ce fait, vous allez écrire :

```
-E100
10D9:0100  55.FD   8B.00   EC.00   56.00   8B.00   76.00   0A.1A
-D100
10D9:0100  FD 00 00 00 00 00 1A 55-8B EC 56 8B 76 0A 8B 04
10D9:0110  8B 76 08 8B 1C 03 C3 8B-76 06 89 04 5E 5D CA 06
10D9:0120  00 .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
```

Est-ce tout ?... Non, il reste encore à indiquer à DEBUG la nouvelle longueur du fichier binaire (cette fois-ci y compris l'en-tête...). Or cette nouvelle longueur est désormais égale à 33 octets (26 + 7) ce qui, en hexadécimal, correspond à la valeur 21. Cette indication figurant dans le registre CX, nous allons donc appeler comme suit ce registre de façon à rajouter 7 à sa valeur actuelle :

```
-R CX
CX 001A
:21
```

Ceci étant, nous avons terminé. Il ne nous reste plus qu'à sauvegarder le fichier binaire modifié et à quitter DEBUG :

```
-W
Ecriture de 0021 octets
-Q
```

Vous voici désormais en possession d'un fichier binaire qui, bien que n'étant pas strictement identique à un fichier binaire produit par l'instruction BSAVE (il eut fallu pour cela répéter en fin de fichier les mêmes 7 codes qu'au début et clôturer le tout par le code de fin de fichier 1A...), n'en demeure pas moins acceptable par l'instruction BLOAD du GW Basic.

Programme d'essai en Basic

La confirmation de ce qui précède va vous être fournie par le programme de démonstration ci-après :

```
10 DEFINT A-Z
20 DIM TABLEAU(12)
30 A=7:B=5:C=0
40 DECALAGE=VARPTR(TABLEAU(0))
50 BLOAD"ADDITION.BIN",DECALAGE
60 ADDITION=VARPTR(TABLEAU(0))
70 CALL ADDITION(A,B,C)
80 PRINT"7+5=";C
```

Si l'on s'intéresse au fonctionnement de ce programme, on remarque que toutes les variables sont de type entier (ligne 10). Ce faisant, la ligne 20 dimensionne un tableau de 26 octets (13 indices, 2 octets par indice) destiné à recevoir les 26 codes-opération de la routine.

Après avoir déclaré toutes les variables à la ligne 30, on recherche à la ligne suivante l'emplacement du début du tableau que l'on enregistre sous la forme d'un DECALAGE au sein du segment du Basic. Notez que la déclaration préalable des variables est indispensable, sans quoi la localisation du tableau est faussée...

Fort de ce DECALAGE, la ligne 50 assure le chargement dans le tableau de l'ensemble des 26 codes-opération de la routine. C'est tout de même préférable (surtout si la routine est longue...) que de procéder octet par octet.

Le tableau étant chargé, il ne reste plus qu'à appeler la routine d'addition au travers d'un CALL ADDITION(A,B,C), en précisant entre parenthèses le nombre de paramètres à lui transmettre. Pour mémoire, il est rappelé que la variable C correspond au résultat de l'addition de A et de B. Vous noterez que la variable ADDITION possède la même valeur de décalage que la variable DECALAGE et qu'il eut été tout aussi possible (mais moins évident...) d'appeler la routine d'addition à l'aide d'un CALL DECALAGE(A,B,C).

Ce programme étant écrit, sauvegardez-le (précaution indispensable si jamais l'ordinateur se plante...) et exécutez-le. Vous devriez voir s'afficher le résultat de l'addition de 7+5, preuve irréfutable du bon fonctionnement de cette seconde méthode d'intégration de routines au Basic interprété.

Récapitulatif de la procédure d'appel

Basic interprété (Basica ou GW Basic) Fichier binaire	
Instruction d'appel	CALL
Paramètres	variables
Transmis sous forme	d'adresses
Empilage dans l'ordre	inverse d'écriture
Procédure de type	FAR
Décalage initial	06H
Pas de décalage	2 octets
Fin de routine	RET n(*)
Directives à spécifier :	aucune

(*) n = Nombre de paramètres transmis X pas de décalage.

BASIC COMPILÉ

Compilateur BASCOM

Pourquoi chercher à compiler un programme écrit en Basic interprété (Basica ou GW Basic) ?... Mais pour en accélérer la vitesse d'exécution !... Sachez, par exemple, que la compilation du Basic avec la version 2.0 du compilateur BASCOM de Microsoft permet d'accroître d'un facteur 4 la vitesse d'exécution d'une application graphique.

Cela étant, comment doit-on procéder pour compiler un programme Basic qui appelle des routines en langage machine ?... La réponse à une telle question ne peut être unique, car il existe autant de procédures distinctes qu'il y a de méthodes d'intégration des routines au langage.

La première procédure que nous allons examiner est celle qui correspond à l'écriture des codes-opération de la routine sous la forme de DATAs, suivie de leur chargement dans un tableau.

Ne tentez surtout pas de compiler tel quel le programme Basic qui utiliserait une telle méthode. Vous n'obtiendriez lors de l'édition de liens qu'un message d'erreur du type :



Cette légère saute d'humeur de l'éditeur de liens provient de l'emploi de l'instruction CALL dans le programme Basic. Avec la méthode d'intégration adoptée, cette instruction CALL ne convient qu'au Basic interprété. Dans le cas d'une version compilable du programme, il vous faut remplacer ce CALL par un CALL ABSOLUTE, comme cela est indiqué dans l'exemple ci-dessous :

```
10 DEFINT A-Z
20 DIM TABLEAU(12)
30 A=7:B=5:C=0:I=0:CODE=0
40 DECALAGE=VARPTR(TABLEAU(0))
50 FOR I=DECALAGE TO DECALAGE+25
60 READ CODE
70 POKE I,CODE
80 NEXT
90 ADDITION=VARPTR(TABLEAU(0))
100 CALL ABSOLUTE(A,B,C,ADDITION)          <--- Ligne à modifier
110 PRINT"7+5=";C
120 END
130 DATA &H55,&H8B,&HEC,&H56,&H8B,&H76,&H0A,&H8B
140 DATA &H04,&H8B,&H76,&H08,&H8B,&H1C,&H03,&HC3
150 DATA &H8B,&H76,&H06,&H89,&H04,&H5E,&H5D,&HCA
160 DATA &H06,&H00
```

Notez au passage la syntaxe un peu particulière du CALL ABSOLUTE qui veut que l'adresse de branchement soit placée entre les parenthèses et non à l'extérieur de celles-ci comme avec un CALL.

La modification du programme ayant été réalisée, il vous faut maintenant le sauvegarder sous forme ASCII (option, A de l'instruction SAVE) et, ceci fait, à procéder à sa compilation et à son édition de liens comme suit :

```
>BASCOM ESSAI/O;
>LINK ESSAI;
```

Vous noterez la présence de l'extension /O à la suite de la commande de compilation. Cette extension a pour effet de créer un fichier exécutable autonome lors de l'édition de liens, c'est-à-dire ne nécessitant pas la présence obligatoire du fichier BRUN20.EXE pour pouvoir être exécuté.

Cette première méthode de compilation convenant aux routines courtes, nous allons maintenant en aborder une seconde plus particulièrement adaptée aux routines longues.

Surtout, n'allez pas croire qu'il vous suffit de compiler tel quel le second programme Basic, celui qui chargeait le fichier binaire de la routine dans un tableau à l'aide d'un BLOAD, pour obtenir un fichier exécutable. Vous seriez gratifié d'un message d'erreur tout aussi incongru qu'indéchiffrable car l'instruction BLOAD qu'emploie ce programme ne peut fonctionner correctement après compilation qu'à la condition d'être précédée par un DEF SEG.

Or, si cela est vrai, par exemple, lors du chargement d'une image (DEF SEG=&HA000 en EGA), cela ne l'est plus pour le programme d'essai qui ne possède aucune définition de segment. Il convient donc de limiter au Basic interprété le chargement d'une routine par l'intermédiaire d'un BLOAD et de lui substituer une toute autre méthode dès que l'on cherche à compiler ce Basic.

Routine en Assembleur

L'approche nouvelle qui doit être faite consiste à intégrer la routine, non plus au sein du programme Basic, mais au contraire à l'extérieur de celui-ci sous la forme d'une librairie intégrable au moment de l'édition de liens.

```
CODE      SEGMENT BYTE 'CODE'  ; Déclaration du segment de code
          PUBLIC  ADDITION     ; Routine de type "public"
          ASSUME  CS:CODE      ; Affectation du segment de code
ADDITION  PROC    FAR          ; Déclaration de procédure FAR
                               ;
          PUSH    BP           ; Sauvegarde pointeur de base
          MOV     BP,SP        ; Transfert pointeur de pile en BP
          PUSH    SI           ; Sauvegarde index source
                               ;
          MOV     SI,(BP+ØAH)  ; Chargement adresse variable A en SI
          MOV     AX,[SI]      ; Transfert valeur variable A en AX
          MOV     SI,(BP+Ø8H)  ; Chargement adresse variable B en SI
          MOV     BX,[SI]      ; Transfert valeur variable B en BX
          ADD     AX,BX        ; Addition de A avec B (résultat dans AX)
          MOV     SI,(BP+Ø6H)  ; Chargement adresse variable C en SI
          MOV     [SI],AX      ; Transfert contenu de AX à la variable C
                               ;
          POP     SI           ; Rétablissement index source
          POP     BP           ; Rétablissement pointeur de base
          RET     6            ; ... et retour à l'envoyeur
                               ;
ADDITION  ENDP                 ; Fin de la procédure d'addition
CODE      ENDS                 ; Fin du segment de code
          END                  ; Fin de la routine
```

Pour ce faire, il vous faut déclarer la routine de type PUBLIC en début du programme source en Assembleur, comme indiqué ci-avant. Sans cette précaution élémentaire, l'éditeur de liens ne saurait retrouver la routine au sein de la librairie et, de ce fait, ne pourrait l'intégrer au fichier exécutable qu'il produit.

Dans le même ordre d'idées, il vous faut également rajouter l'indication 'CODE' à la suite de la déclaration du segment de code de façon à ce que l'éditeur de liens sache où le situer.

Le programme source de la routine ayant été modifié comme indiqué, vous allez procéder à son assemblage en tapant :

```
>MASM ADDITION;
```

Vous obtenez ainsi un fichier objet (extension .OBJ) propre à être incorporé au fichier exécutable lors de l'édition de liens.

Programme d'essai en Basic

L'opération d'assemblage étant terminée, il vous faut maintenant aborder l'écriture du programme Basic d'appel de la routine. Par rapport aux versions précédentes, vous allez découvrir l'extrême simplification qu'autorise cette méthode d'intégration, puisque le programme se résume à ces quelques lignes :

```
10 DEFINT A-Z
20 A=7:B=5:C=0
30 CALL ADDITION(A,B,C)
40 PRINT"7+5=";C
```

Ici, point de déclaration de tableau ni de recherche d'adresse de début qu'il faut attribuer à la constante ADDITION. Tout se résume à un simple CALL suivi du nom de la routine et de ses paramètres. C'est dire le dépouillement...

Notez tout de même qu'il faut attribuer une valeur aux variables A, B et C avant l'appel de la routine de façon à fixer leur adresse en mémoire. Sans cela, vous auriez quelques surprises...

Procédure d'intégration au Basic

Reste encore à vérifier que tout cela fonctionne !... Pour ce faire, vous allez compiler ce programme d'ESSAI, procéder à son édition de liens et l'exécuter :

```
>BASCOM ESSAI/O;

>LINK ESSAI ADDITION;
```

Notez au passage la procédure un peu particulière de l'éditeur de liens Microsoft qui autorise l'intégration d'une routine .OBJ, pourvu que le nom de cette routine soit spécifié à la suite du nom du fichier objet sur la même ligne de commande.

Il existe une autre procédure qui consiste à créer une librairie .LIB à partir de la routine .OBJ à l'aide de l'utilitaire LIB livré avec le Macro-Assembleur et à l'incorporer comme suit :

```
>LIB

Microsoft (R) Library Manager  Version 3.02
Copyright (C) Microsoft Corp 1983, 1984, 1985.  All rights reserved.

Library name: ADDITION
Library does not exist.  Create? Y
Operations: +ADDITION
List file:
```

Après avoir lancé l'utilitaire LIB, vous lui précisez le nom de la librairie que vous voulez créer (ADDITION.LIB dans notre cas...) et vous lui demandez d'y inclure le fichier objet de la routine d'addition. Ce faisant, vous obtenez une librairie dénommée ADDITION.LIB que vous pouvez alors incorporer comme suit à l'éditeur de liens :

```
>LINK ESSAI

Microsoft (R) Overlay Linker  Version 3.06
Copyright (C) Microsoft Corp 1983, 1984, 1985, 1986. All rights reserved.

Run File [ESSAI.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]: ADDITION       <--- Intégration de la librairie.
```

De ce qui précède, on constate qu'il n'existe pas de solution universelle en matière d'écriture d'un programme qui puisse à la fois fonctionner en Basic interprété et en compilé.

La solution, quand on décide de développer en Basic compilé, consiste à incorporer ses routines dans un tableau à l'aide d'un BLOAD pendant la phase de développement, pour les lier ensuite au module exécutable par le biais d'une librairie lors de la phase finale de compilation, avec ce que cela sous-entend de modifications à apporter au programme source pour le rendre compilable.

Récapitulatif de la procédure d'appel

Basic compilé (Compilateur BASCOM)	
Instruction d'appel	CALL (routine externe) CALL ABSOLUTE (code en tableau)
Paramètres	variables
Transmis sous forme	d'adresses
Empilage dans l'ordre	inverse d'écriture
Procédure de type	FAR
Décalage initial	06H
Pas de décalage	2 octets
Fin de routine	RET n(*)
Directives à spécifier	PUBLIC et 'CODE' (routines externes uniquement)

(*) n = Nombre de paramètres transmis × pas de décalage.

QUICK BASIC version 2.0

Sous-programme

Apparu en 1987 sur le marché des langages de programmation, le Quick Basic de Microsoft est un Basic compilé qui se prête bien à l'intégration de routines écrites en Assembleur, dans la mesure où ce langage est systématiquement compilé chaque fois que l'on en lance l'exécution (il ne possède pas d'interpréteur...), évitant ainsi au programmeur de devoir écrire deux versions différentes du programme selon qu'il est en phase de développement ou en phase finale de mise en forme du module exécutable.

Routine en Assembleur

Sur le plan de la structure des routines adaptées au Quick Basic, sachez que celle-ci ne diffère en rien de celle du Basic compilé, à savoir que les paramètres sont transmis sous forme d'adresses (on ne transmet pas la valeur du paramètre mais l'adresse où il se trouve...) et que le RET final se doit de tenir compte du nombre d'octets correspondant aux paramètres.

```
;               ****************************************
;               Routine d'addition adaptée au QuickBasic
;               ****************************************

CODE      SEGMENT BYTE 'CODE'   ; Déclaration du segment de code
          PUBLIC  ADDITION      ; Routine de type "public"
          ASSUME  CS:CODE       ; Affectation du segment de code
ADDITION  PROC    FAR           ; Déclaration de procédure FAR
                                ;
          PUSH    BP            ; Sauvegarde pointeur de base
          MOV     BP,SP         ; Transfert pointeur de pile en BP
          PUSH    SI            ; Sauvegarde index source
                                ;
          MOV     SI,(BP+0AH)   ; Chargement adresse variable A en SI
          MOV     AX,[SI]       ; Transfert valeur variable A en AX
          MOV     SI,(BP+08H)   ; Chargement adresse variable B en SI
          MOV     BX,[SI]       ; Transfert valeur variable B en BX
          ADD     AX,BX         ; Addition de A avec B (résultat dans AX)
          MOV     SI,(BP+06H)   ; Chargement adresse variable C en SI
          MOV     [SI],AX       ; Transfert contenu de AX à la variable C
                                ;
          POP     SI            ; Rétablissement index source
          POP     BP            ; Rétablissement pointeur de base
          RET     6             ; ... et retour à l'envoyeur
                                ;
ADDITION  ENDP                  ; Fin de la procédure d'addition
CODE      ENDS                  ; Fin du segment de code
          END                   ; Fin de la routine
```

Après déclaration et affectation du segment de code (c'est lui qui renferme le code objet de la routine...), la routine d'addition est déclarée en procédure FAR.

Ceci étant, les lignes suivantes sauvegardent l'index source SI (on eut pu tout aussi bien employer l'index destinataire DI...) et le pointeur de base BP avant que ne soit attribué à ce dernier la valeur du pointeur de pile SP. Or, c'est cette valeur qui, mise dans BP, va nous servir à gérer les paramètres dans la pile.

Vous noterez que l'on ne transmet pas à la routine les valeurs numériques des paramètres, mais bien au contraire l'adresse où l'on est susceptible de les trouver et que ceux-ci s'empilent dans l'ordre inverse de leur écriture, le dernier paramètre transmis étant affecté du décalage minimum de 06. Ceci explique pourquoi le transfert des paramètres dans les registres AX et BX se passe en deux temps, l'index source recevant l'adresse dans un premier temps, adresse qui sert ensuite à transférer la valeur des paramètres d'entrée dans les registres AX et BX.

Dans ces conditions, le pas de décalage employé au sein de la routine pour lire la pile est de 2 octets, que les paramètres lui soient transmis (et donc les variables employées...) sous la forme d'entiers sur deux octets ou de type simple ou double précision. Ne tombez pas dans le piège qui consiste à croire que le pas de décalage est dépendant du type de variable choisie : il n'en est rien !...

L'addition étant effectuée et le résultat expédié vers la variable C, il ne reste plus qu'à rétablir dans leur valeur d'origine l'index source et le pointeur de base avant de rendre la main au programme d'appel. Notez au passage que le RET final est suivi du nombre 6, ce qui équivaut à 3 paramètres de 2 octets chacun et assure la gestion correcte de la pile.

Vous remarquerez également que la routine doit être déclarée de type « PUBLIC » afin de pouvoir être acceptée par l'éditeur de liens. Dans le même ordre d'idées, il vous faut également rajouter l'indication 'CODE' à la suite de la déclaration du segment de code de façon à ce que l'éditeur de liens sache où le situer.

Procédure d'intégration au QuickBasic 2.0

La routine d'addition ayant été assemblée, vous allez pouvoir procéder à son intégration au sein du Quick Basic. Pour ce faire, vous allez devoir créer une librairie dénommée USERLIB.EXE (ou tout autre nom de votre choix, USERLIB étant l'appellation par défaut...) à l'aide de l'utilitaire BUILDLIB en tapant :

```
>BUILDLIB ADDITION;
```

Ceci fait, il vous suffit alors de procéder au chargement de cette librairie en incluant l'option L (pour librairie...) à la suite de l'ordre de lancement du Quick Basic :

```
>QB/L
```

...un point, c'est tout !

Programme d'essai en Quick Basic

La librairie étant désormais chargée en même temps que le Quick Basic, il ne vous reste plus qu'à écrire le petit programme d'essai que voici pour vérifier si l'appel de la routine d'addition s'effectue correctement :

```
DEFINT A-Z
A=7:B=5:C=Ø
CALL Addition(A,B,C)
PRINT"7+5=";C
```

Si vous êtes curieux, amusez-vous à remplacer les paramètres A et B par leurs valeurs numériques lors de l'appel de la routine d'addition. Vous allez découvrir que le Quick Basic ne fait aucune différence et délivre un résultat exact, que les paramètres soient transmis sous la forme de variables ou de valeurs numériques. Génial, n'est-ce pas ?

Quant aux différences d'écriture des noms de routines écrits en majuscules (Assembleur) ou en minuscules (Quick Basic), sachez qu'elles sont sans importance car ce dernier traite indifféremment majuscules et minuscules, autorisant ainsi quelques fantaisies dans la présentation des programmes.

La procédure d'intégration de routines pendant la phase de développement du programme étant acquise, il vous faut maintenant aborder celle qui a trait à la mise en forme du module exécutable autonome final.

Un module exécutable autonome s'obtient en spécifiant l'option /O lors de la compilation, l'édition des liens s'effectuant alors avec la librairie BCOM20.LIB et ce, de manière implicite sans qu'il soit nécessaire de préciser le moindre nom de librairie. La seule précaution à observer consiste à placer cette librairie dans votre sous-répertoire de travail.

```
>QB ESSAI/O;
```

Quant à l'intégration de la routine d'addition, celle-ci s'accomplit lors de l'édition de liens. Il suffit pour cela de préciser le nom du fichier objet de la routine à la suite du nom du programme auquel on souhaite l'intégrer et le tour est joué :

```
>LINK ESSAI ADDITION;
```

En conclusion, le Quick Basic (tout du moins dans sa version 2.0...) se prête admirablement bien à l'intégration de routines écrites en Assembleur, que cela soit en phase de développement ou en phase finale de mise en forme du module exécutable. La procédure est simple, le programme ainsi que les routines n'ont pas besoin d'être réécrits pour passer d'une phase à l'autre.

Récapitulatif de la procédure d'appel

Quick Basic version 2.0 Sous-programme		
Instruction d'appel		CALL
Paramètres		variables et valeurs numériques
Transmis sous forme		d'adresses
Empilage dans l'ordre		inverse d'écriture
Procédure de type		FAR
Décalage initial		06H
Pas de décalage		2 octets
Fin de routine		RET n(*)
Directives à spécifier	:	PUBLIC et 'CODE'

(*) n = Nombre de paramètres transmis × pas de décalage.

QUICK BASIC version 4.0

Sous-programme

Apparue en 1988 sur le marché des langages de programmation, cette version 4.0 offre la possibilité d'appeler une routine Assembleur comme s'il s'agissait d'un sous-programme ou d'une fonction. Nous allons débuter par la procédure d'intégration d'une routine en tant que sous-programme qui, bien qu'identique en l'esprit, se distingue de celle des versions précédentes par une multitude de petits détails.

Routine en Assembleur

Sur le plan de la structure, une routine appelable par le Quick Basic 4.0 en tant que sous-programme ne diffère en rien de celle employée avec les versions précédentes. Sachez que les paramètres sont transmis sous forme d'adresses (on ne transmet pas la valeur du paramètre mais l'adresse où il se trouve...) et que le RET final se doit de tenir compte du nombre d'octets correspondant aux paramètres.

Afin d'illustrer ce qui va suivre, nous avons choisi une routine d'addition de 2 nombres entiers A et B, le résultat étant placé dans la variable C (également entière).

```
;             *****************************************
;             Routine d'addition adaptée au QuickBasic 4.Ø
;                            Sous-programme
;             *****************************************

CODE          SEGMENT BYTE 'CODE'   ; Déclaration du segment de code.
              ASSUME  CS:CODE       ; Routine de type "public".
              PUBLIC  ADDITION      ; Affectation du segment de code.
ADDITION      PROC    FAR           ; Procédure de type FAR.
                                    ;
              PUSH    BP            ; Sauvegarde pointeur de base.
              MOV     BP,SP         ; Transfert pointeur de pile en BP.
              PUSH    SI            ; Sauvegarde index source.
                                    ;
              MOV     SI,(BP+ØAH)   ; Chargement adresse de A en SI.
              MOV     AX,[SI]       ; Transfert valeur de A en AX.
              MOV     SI,(BP+Ø8H)   ; Chargement adresse de B en SI.
              MOV     BX,[SI]       ; Transfert valeur de B en BX.
              ADD     AX,BX         ; Addition de A+B (résultat en AX).
              MOV     SI,(BP+Ø6H)   ; Chargement adresse de C en SI.
              MOV     [SI],AX       ; Transfert contenu de AX à la variable C.
                                    ;
              POP     SI            ; Rétablissement index source.
              POP     BP            ; Rétablissement pointeur de base
              RET     6             ; ... et rerour à l'envoyeur.
                                    ;
ADDITION      ENDP                  ; Fin de la procédure d'addition.
CODE          ENDS                  ; Fin du segment de code.
              END                   ; Fin de la routine.
```

Après déclaration et affectation du segment de code (c'est lui qui renferme le code objet de la routine...), la routine d'addition est déclarée en procédure de type FAR.

Ceci étant, les lignes suivantes sauvegardent l'index source SI (on eut pu tout aussi bien employer l'index destinataire DI...) et le pointeur de base BP avant que ne soit attribué à ce dernier la valeur du pointeur de pile SP. Or, c'est cette valeur qui, mise dans BP, va nous servir à gérer les paramètres dans la pile.

Vous noterez que l'on ne transmet pas à la routine les valeurs numériques des paramètres, mais bien au contraire l'adresse où l'on est susceptible de les trouver et que ceux-ci s'empilent dans l'ordre inverse de leur écriture, le

dernier paramètre transmis étant affecté du décalage minimum de 06. Ceci explique pourquoi le transfert des paramètres dans les registres AX et BX se passe en deux temps, l'index source recevant l'adresse dans un premier temps, adresse qui sert ensuite à transférer la valeur des paramètres d'entrée dans les registres AX et BX dans un second temps.

Dans ces conditions, le pas de décalage employé au sein de la routine pour lire la pile est de 2 octets, que les paramètres lui soient transmis (et donc les variables employées...) sous la forme d'entiers sur deux octets ou de type simple ou double précision. Ne tombez pas dans le piège qui consiste à croire que le pas de décalage est dépendant du type de variable choisie : il n'en est rien !...

L'addition étant effectuée et le résultat expédié vers la variable C, il ne reste plus qu'à rétablir dans leur valeur d'origine l'index source et le pointeur de base avant de rendre la main au programme d'appel. Notez au passage que le RET final est suivi du nombre 6, ce qui équivaut à 3 paramètres de 2 octets chacun et assure la gestion correcte de la pile.

Vous remarquerez également que la routine doit être déclarée de type PUBLIC afin de pouvoir être reconnue par l'éditeur de liens. Dans le même ordre d'idées, il vous faut rajouter l'indication 'CODE' à la suite de la déclaration du segment de code de façon à ce que l'éditeur de liens sache où le situer.

Procédure d'intégration au Quick Basic 4.0

La routine étant tapée sous un éditeur, vous allez l'assembler avec le Macro-Assembleur MASM de Microsoft :

```
>MASM ADDITION;
```

Ceci fait, il vous faut maintenant l'incorporer à l'environnement intégré de développement du Quick Basic de façon à pouvoir l'appeler directement à partir de votre programme.

Pour ce faire, il vous faut créer deux librairies de développement, l'une d'extension. QLB qui sera appelée à chaque compilation et édition de liens déclenchées par l'option RUN de l'environnement intégré, l'autre d'extension .LIB qui permet d'obtenir un fichier exécutable autonome.

La première librairie, baptisée ESSAI.QLB pour la circonstance (bien que tout autre nom de votre choix convienne...), s'obtient en lançant comme suit l'éditeur de liens :

```
>LINK /Q ADDITION.OBJ,ESSAI.QLB,,BQLB4Ø.LIB;
```

La seconde librairie, baptisée ESSAI.LIB (le nom doit être le même que celui de la première librairie, hormis l'extension qui diffère...) s'obtient, quant à elle, avec l'utilitaire LIB de Microsoft :

```
>LIB ESSAI.LIB+ADDITION.OBJ;
```

Important : *Il est à noter que la création de ces 2 librairies ne peut en aucun cas être exécutée à partir de l'environnement intégré du Quick Basic, même si l'option « MAKE Library » peut le laisser croire...*

Ceci fait, il vous suffit maintenant de procéder au chargement de la librairie ESSAI.QLB en même temps que le Quick Basic en incluant l'option / L (pour librairie...), suivie du nom complet de la librairie à la suite de l'ordre de lancement du Quick Basic :

```
>QB/L ESSAI.QLB
```

Programme d'essai en Quick Basic

La librairie étant désormais chargée en même temps que le Quick Basic, il ne vous reste plus qu'à écrire le petit programme d'essai que voici pour vérifier si l'appel de la routine d'addition s'effectue correctement :

```
DEFINT A-Z
A=7:B=5:C=Ø
CALL Addition(A,B,C)
PRINT"7+5=";C
```

Si vous êtes curieux, amusez-vous à remplacer les paramètres A et B par leurs valeurs numériques lors de l'appel de la routine d'addition. Vous allez découvrir que le Quick Basic ne fait aucune différence et délivre un résultat exact, que les paramètres soient transmis sous la forme de variables ou de valeurs numériques. Génial, n'est-ce pas ?

Quant aux différences d'écriture des noms de routines écrits en majuscules (Assembleur) ou en minuscules (Quick Basic), sachez qu'elles sont sans importance car ce dernier traite indifféremment majuscules et minuscules, autorisant ainsi quelques fantaisies dans la présentation des programmes.

La procédure d'intégration de routines au sein de l'environnement intégré du QuickBasic étant acquise, il reste à aborder celle qui a trait à la mise en forme du module exécutable autonome.

Pour y parvenir, point n'est besoin de sortir de l'environnement intégré. Il vous suffit de sélectionner l'option **MAKE EXE File** et de préciser, lorsqu'on vous le demande, que vous désirez un fichier exécutable autonome (Stand-Alone EXE File pour la version anglaise...), un point c'est tout... !

Récapitulatif de la procédure d'appel

Quick Basic version 4.0 Sous-programme	
Instruction d'appel	CALL
Paramètres	variables et valeurs numériques
Transmis sous forme	d'adresses
Empilage dans l'ordre	inverse d'écriture
Procédure de type	FAR
Décalage initial	06H
Pas de décalage	2 octets
Fin de routine	RET n(*)
Directives à spécifier	PUBLIC et 'CODE'

(*) n = Nombre de paramètres transmis × pas de décalage.

QUICK BASIC version 4.0

Fonction

Pouvoir assimiler une routine Assembleur à une fonction est une nouveauté (pour les Basics, s'entend...) bien pratique qu'offre la version 4.0 de Quick Basic. Toutefois, sa mise en œuvre s'accompagne de déclarations nouvelles et de modifications d'écriture des routines et des programmes d'appel qu'il convient d'examiner dans le détail.

Routine en Assembleur

Sur le plan de la structure, une routine considérée comme une fonction réexpédie automatiquement en fin d'exécution le contenu du registre AX vers le programme appelant. En conséquence, il n'y a pas lieu de prévoir, comme précédemment, le retour des informations vers la routine puisque la procédure d'appel s'en charge d'elle-même.

Afin d'illustrer ce qui va suivre, nous avons choisi une routine d'addition de 2 nombres entiers A et B, le résultat étant attribué à la variable entière C (qui n'est pas transmise à la routine, rappelons-le, puisqu'il s'agit d'une fonction...).

```
;          ****************************************************
;          Routine d'addition adaptée au QuickBasic 4.Ø
;                             Fonction
;          ****************************************************

CODE       SEGMENT BYTE 'CODE'  ; Déclaration du segment de code.
           ASSUME  CS:CODE      ; Routine de type "public".
           PUBLIC  ADDITION     ; Affectation du segment de code.
ADDITION   PROC    FAR          ; Procédure de type FAR.
                                ;
           PUSH    BP           ; Sauvegarde pointeur de base.
           MOV     BP,SP        ; Transfert pointeur de pile en BP.
           PUSH    SI           ; Sauvegarde index source.
                                ;
           MOV     SI,(BP+Ø8H)  ; Chargement adresse de A en SI.
           MOV     AX,[SI]      ; Transfert valeur de A en AX.
           MOV     SI,(BP+Ø6H)  ; Chargement adresse de B en SI.
           MOV     BX,[SI]      ; Transfert valeur de B en BX.
           ADD     AX,BX        ; Addition de A+B (résultat en AX).
                                ;
           POP     SI           ; Rétablissement index source.
           POP     BP           ; Rétablissement pointeur de base
           RET     4            ; ... et retour à l'envoyeur.
                                ;
ADDITION   ENDP                 ; Fin de la procédure d'addition.
CODE       ENDS                 ; Fin du segment de code.
           END                  ; Fin de la routine.
```

Après déclaration et affectation du segment de code (c'est lui qui renferme le code objet de la routine...), la routine d'addition est déclarée en procédure de type FAR.

Ceci étant, les lignes suivantes sauvegardent l'index source SI (on eut pu tout aussi bien employer l'index destinataire DI...) et le pointeur de base BP avant que ne soit attribué à ce dernier la valeur du pointeur de pile SP. Or, c'est cette valeur qui, mise dans BP, va nous servir à gérer les paramètres dans la pile.

Vous noterez que l'on ne transmet pas à la routine les valeurs numériques des paramètres, mais bien au contraire l'adresse où l'on est susceptible de les trouver et que ceux-ci s'empilent dans l'ordre inverse de leur écriture, le dernier paramètre transmis étant affecté du décalage minimum de 06. Ceci

explique pourquoi le transfert des paramètres dans les registres AX et BX se passe en deux temps, l'index source recevant l'adresse dans un premier temps, adresse qui sert ensuite à transférer la valeur des paramètres d'entrée dans les registres AX et BX dans un second temps.

Dans ces conditions, le pas de décalage employé au sein de la routine pour lire la pile est de 2 octets, que les paramètres lui soient transmis (et donc les variables employées...) sous la forme d'entiers sur deux octets ou de type simple ou double précision. Ne tombez pas dans le piège qui consiste à croire que le pas de décalage est dépendant du type de variable choisie : il n'en est rien !...

L'addition étant effectuée et le résultat expédié automatiquement par la procédure d'appel vers la variable C, il ne reste plus qu'à rétablir dans leur valeur d'origine l'index source et le pointeur de base avant de rendre la main au programme d'appel. Notez au passage que le RET final est suivi du nombre 4, ce qui équivaut à 2 paramètres de 2 octets chacun et assure la gestion de la pile.

Vous remarquerez également que la routine doit être déclarée de type PUBLIC afin de pouvoir être reconnue par l'éditeur de liens. Dans le même ordre d'idées, il vous faut rajouter l'indication 'CODE' à la suite de la déclaration du segment de code de façon à ce que l'éditeur de liens sache où le situer.

Procédure d'intégration au Quick Basic 4.0

La routine étant tapée sous un éditeur, vous allez l'assembler avec le Macro-Assembleur MASM de Microsoft :

```
>MASM ADDITION;
```

Ceci fait, il vous faut maintenant l'incorporer à l'environnement intégré de développement du Quick Basic de façon à pouvoir l'appeler directement à partir de votre programme.

Pour ce faire, il vous faut créer deux librairies de développement, l'une d'extension .QLB qui sera appelée à chaque compilation et édition de liens déclenchées par l'option RUN de l'environnement intégré, l'autre d'extension .LIB qui permet d'obtenir un fichier exécutable autonome.

La première librairie, baptisée ESSAI.QLB pour la circonstance (bien que tout autre nom de votre choix convienne...), s'obtient en lançant comme suit l'utilitaire BQLB40.LIB :

```
>LINK /Q ADDITION.OBJ,ESSAI.QLB,,BQLB40.LIB;
```

La seconde librairie, baptisée ESSAI.LIB (le nom doit être le même que celui de la première librairie, hormis l'extension qui diffère...) s'obtient, quant à elle, avec l'utilitaire LIB de Microsoft :

```
>LIB ESSAI.LIB+ADDITION.OBJ;
```

Important : *Il est à noter que la création de ces 2 librairies ne peut en aucun cas être exécutée à partir de l'environnement intégré du Quick Basic, même si l'option « MAKE Library » peut le laisser croire...*

Ceci fait, il vous suffit maintenant de procéder au chargement de la librairie ESSAI.QLB en même temps que le Quick Basic en incluant l'option / L (pour librairie...), suivie du nom complet de la librairie à la suite de l'ordre de lancement du Quick Basic :

```
>QB/L ESSAI.QLB
```

Programme d'essai en Quick Basic

La librairie étant désormais chargée en même temps que le Quick Basic, il ne vous reste plus qu'à écrire le petit programme d'essai que voici pour vérifier si l'appel de la routine d'addition s'effectue correctement :

```
DEFINT A-Z
DECLARE FUNCTION Addition (A, B)
A = 5: B = 7: C = Ø
C = Addition(A, B)
PRINT "5+7="; C
```

Vous noterez la déclaration de la routine en tant que fonction, déclaration qui s'opère avec l'instruction DECLARE FUNCTION suivie du nom de la routine et des paramètres qu'on lui transmet. En l'occurence, ceux-ci sont de type entier puisqu'ils ont été préalablement déclarés comme tel avec un DEFINT.

Si vous êtes curieux, amusez-vous à remplacer les paramètres A et B par leurs valeurs numériques lors de l'appel de la routine d'addition. Vous allez découvrir que le Quick Basic ne fait aucune différence et délivre un résultat exact, que les paramètres soient transmis sous la forme de variables ou de valeurs numériques. Génial, n'est-ce pas ?

Quant aux différences d'écriture des noms de routines écrits en majuscules (Assembleur) ou en minuscules (Quick Basic), sachez qu'elles sont sans importance car ce dernier traite indifféremment majuscules et minuscules, autorisant ainsi quelques fantaisies dans la présentation des programmes.

La procédure d'intégration de routines au sein de l'environnement intégré du Quick Basic étant acquise, il reste à aborder celle qui a trait à la mise en forme du module exécutable autonome.

Pour y parvenir, point n'est besoin de sortir de l'environnement intégré. Il vous suffit de sélectionner l'option **MAKE EXE File** et de préciser, lorsqu'on vous le demande, que vous désirez un fichier exécutable autonome (Stand-Alone EXE File pour la version anglaise...), un point c'est tout... !

Récapitulatif de la procédure d'appel

Quick Basic version 4.0 Fonction		
Instruction d'appel	:	aucune
Paramètres	:	variables et valeurs numériques
Transmis sous forme		d'adresses
Empilage dans l'ordre	:	inverse d'écriture
Procédure de type	:	FAR
Décalage initial	:	06H
Pas de décalage		2 octets
Fin de routine		RET n(*)
Directives à spécifier		PUBLIC et 'CODE'

(*) n = Nombre de paramètres transmis X pas de décalage.

TURBO BASIC version 1.0

Code en ligne

Le Turbo Basic Borland est un langage compilé qui se prête fort bien à l'intégration de routines écrites en Assembleur. Ses procédures d'intégration sont toutefois assez inhabituelles, notamment celle qui consiste à écrire les codes-opération de chaque routine au sein de sous-programmes de type INLINE.

Routine en Assembleur

Afin d'illustrer ce qui va suivre, nous avons choisi une routine d'addition de 2 nombres entiers qui, bien que totalement inutile puisque le Turbo Basic possède déjà cela, va nous permettre de contrôler la transmission correcte des paramètres entre le programme et la routine.

```
        ****************************************
        Routine d'addition adaptée au TurboBasic
        ****************************************

CODE     SEGMENT BYTE         ; Déclaration du segment de code
         ASSUME  CS:CODE      ; Affectation du segment de code
ADDITION PROC    FAR          ; Déclaration de procédure FAR
                              ;
         PUSH    BP           ; Sauvegarde pointeur de base
         MOV     BP,SP        ; Transfert pointeur de pile en BP
         PUSH    SI           ; Sauvegarde index source
                              ;
         MOV     SI,(BP+ØEH)  ; Chargement adresse variable A en SI
         MOV     AX,[SI]      ; Transfert valeur variable A en AX
         MOV     SI,(BP+ØAH)  ; Chargement adresse variable B en SI
         MOV     BX,[SI]      ; Transfert valeur variable B en BX
         ADD     AX,BX        ; Addition de A et de B (résultat dans AX)
         MOV     SI,(BP+Ø6H)  ; Chargement adresse variable C en SI
         MOV     [SI],AX      ; Transfert contenu de AX à la variable C
                              ;
         POP     SI           ; Rétablissement index source
         POP     BP           ; Rétablissement pointeur de base
                              ;
ADDITION ENDP                 ; Fin de la procédure d'addition
CODE     ENDS                 ; Fin du segment de code
         END                  ; Fin de la routine
```

Après déclaration et affectation du segment de code (c'est lui qui renferme le code objet de la routine...), la routine d'addition est déclarée en procédure FAR.

Ceci étant, les lignes suivantes sauvegardent l'index source SI (on eut pu tout aussi bien employer l'index destinataire DI...) et le pointeur de base BP avant que ne soit attribué à ce dernier la valeur du pointeur de pile SP. Or, c'est cette valeur qui, mise dans BP, va nous servir à gérer les paramètres dans la pile.

Vous noterez que l'on ne transmet pas à la routine les valeurs numériques des paramètres, mais bien au contraire l'adresse où l'on est susceptible de les trouver et que ceux-ci s'empilent dans l'ordre inverse de leur écriture, le dernier paramètre transmis étant affecté du décalage minimum de 06. Ceci explique pourquoi le transfert des paramètres dans les registres AX et BX se passe en deux temps, l'index source recevant l'adresse dans un premier temps, adresse qui sert ensuite à transférer la valeur des paramètres d'entrée dans les registres AX et BX dans un second temps.

Les adresses étant transmises sous la forme segment + décalage, il s'en suit que le pas de décalage employé au sein de la routine pour lire la pile est de 4 octets, que les paramètres lui soient transmis (et donc les variables employées...) sous la forme d'entiers sur deux octets ou de type simple ou double précision. Ne tombez pas dans le piège qui consiste à croire que le pas de décalage est dépendant du type de variable choisie : il n'en est rien !...

L'addition étant effectuée et le résultat expédié vers la variable C, il ne reste plus qu'à rétablir dans leur valeur d'origine l'index source et le pointeur de base avant de rendre la main au programme d'appel. Notez au passage l'absence de RET final (non, ce n'est pas un oubli...). Le Turbo Basic est ainsi fait qu'il gère lui-même la pile au retour de la routine, ce qui oblige le programmeur à modifier quelque peu ses habitudes...

Procédure d'intégration au Turbo Basic

Ainsi, lorsque le nombre de codes-opération de la routine est réduit, vous allez incorporer ces derniers au sein du programme Basic, non pas à la suite de DATAs (bien que cela soit possible...), mais bien au contraire au sein d'un sous-programme de type INLINE.

La difficulté, c'est de déterminer les codes-opération de la routine (il n'est pas donné à tout le monde de savoir écrire directement en langage machine...). Pour ce faire, il vous faut commencer par assembler le code source de la routine à l'aide d'un Macro-Assembleur (en l'occurence il s'agit de celui de Microsoft...) de façon à obtenir un fichier.OBJ :

```
>MASM ADDITION;
```

A partir de ce fichier objet, vous allez produire un fichier exécutable d'extension .EXE en lançant comme suit l'édition de liens :

```
>LINK ADDITION;
```

Ne vous préoccupez pas du message d'avertissement « Warning : no stack segment » que va vous délivrer l'éditeur de liens. Cela tient, comme vous l'aviez deviné, à l'absence de segment de pile au sein de la routine. Or, comme ce segment n'est pas indispensable...

Ceci étant, vous allez maintenant convertir ce fichier exécutable en un fichier binaire d'extension .BIN en lançant comme suit l'utilitaire de conversion EXE2BIN livrée avec le MS DOS de la machine :

```
>EXE2BIN ADDITION
```

Vous obtenez alors un fichier, ADDITION.BIN qu'il vous suffit de désassembler avec l'utilitaire DEBUG du MSDOS pour obtenir les codes-opération de la routine :

```
C:\QBASIC>DEBUG ADDITION.BIN
-U
11F2:Ø1ØØ 55            PUSH    BP
11F2:Ø1Ø1 8BEC          MOV     BP,SP
11F2:Ø1Ø3 56            PUSH    SI
11F2:Ø1Ø4 8B76ØE        MOV     SI,[BP+ØE]
11F2:Ø1Ø7 8BØ4          MOV     AX,[SI]
11F2:Ø1Ø9 8B76ØA        MOV     SI,[BP+ØA]
11F2:Ø1ØC 8B1C          MOV     BX,[SI]
11F2:Ø1ØE Ø3C3          ADD     AX,BX
11F2:Ø11Ø 8B76Ø6        MOV     SI,[BP+Ø6]
11F2:Ø113 89Ø4          MOV     [SI],AX
11F2:Ø115 5E            POP     SI
11F2:Ø116 5D            POP     BP
-Q
```

Programme d'essai en Turbo Basic

En possession des codes-opération de la routine d'addition, vous allez incorporer ces derniers au sein du programme Basic, non pas à la suite de DATAs (bien que cela soit possible...), mais bien au contraire au sein d'un sous-programme de type INLINE comme dans l'exemple suivant :

```
DEFINT A-Z
A=5:B=7:C=Ø
CALL Addition(A,B,C)
PRINT "5+7=";C                    ------ Attention: ceci est une apostrophe
END                               |                 et non une virgule !
                                  |
SUB Addition INLINE               v
    $INLINE &H55               ' PUSH    BP
    $INLINE &H8B,&HEC          ' MOV     BP,SP
    $INLINE &H56               ' PUSH    SI
    $INLINE &H8B,&H76,&HØE     ' MOV     SI,(BP+ØE)
    $INLINE &H8B,&HØ4          ' MOV     AX,[SI]
    $INLINE &H8B,&H76,&HØA     ' MOV     SI,(BP+ØA)
    $INLINE &H8B,&H1C          ' MOV     BX,[SI]
    $INLINE &HØ3,&HC3          ' ADD     AX,BX
    $INLINE &H8B,&H76,&HØ6     ' MOV     SI,(BP+Ø6)
    $INLINE &H89,&HØ4          ' MOV     [SI],AX
    $INLINE &H5E               ' POP     SI
    $INLINE &H5D               ' POP     BP
END SUB
```

Vous noterez au travers de cet exemple que les codes-opération figurent sous leur forme hexadécimale à la suite d'instructions $INLINE contenues dans un sous-programme de type INLINE. Bien entendu, mais vous l'aviez deviné, la portion désassemblée de la routine est totalement superflue. Elle ne figure en temps que REMs (à la suite d'apostrophes...) que pour mieux expliquer certaines particularités d'écriture.

Quant au reste du programme en Turbo Basic, il est on ne peut plus dépouillé. Après avoir déclaré les variables comme devant être entières et leur avoir attribué une valeur, la routine est appelée au travers d'un CALL, comme s'il s'agissait d'un sous-programme quelconque.

Cela étant, il ne vous reste plus qu'à exécuter ce petit programme d'essai afin de vous assurer que cette procédure d'intégration dite du « code en ligne » fonctionne correctement.

Pour le reste, à savoir la production d'un module exécutable autonome du programme Basic intégrant une (ou plusieurs...) routines, il suffit de choisir l'option « compile to EXE file » dans le menu «Options » pour que, lors de la compilation, le TurboBasic dépose sur la disquette ou le disque dur un module exécutable autonome.

Récapitulatif de la procédure d'appel

Turbo Basic version 1.0 Code en ligne	
Instruction d'appel	CALL
Paramètres	variables
Transmis sous forme	d'adresses
Empilage dans l'ordre	inverse d'écriture
Procédure de type	FAR
Décalage initial	06H
Pas de décalage	4 octets
Fin de routine	pas de RET
Directives à spécifier	aucune

TURBO BASIC Version 1.0

Fichier .COM

La méthode d'intégration précédente n'étant valable que pour des routines courtes (allez donc recopier ainsi plusieurs centaines d'octets...), en voici une seconde qui consiste à transformer la routine en un fichier binaire d'extension .COM et à l'appeler au travers d'un sous-programme de type INLINE.

Routine en Assembleur

Afin d'illustrer ce qui va suivre, nous avons choisi une routine d'addition de 2 nombres entiers qui, si elle n'est pas d'une nécessité impérative (le Turbo Basic en est déjà pourvu...), va nous permettre de contrôler la transmission correcte des paramètres entre le programme appelant et la routine.

```
            ******************************************
            Routine d'addition adaptée au TurboBasic
            ******************************************

CODE        SEGMENT BYTE         ; Déclaration du segment de code
            ASSUME  CS:CODE      ; Affectation du segment de code
ADDITION    PROC    FAR          ; Déclaration de procédure FAR
                                 ;
            PUSH    BP           ; Sauvegarde pointeur de base
            MOV     BP,SP        ; Transfert pointeur de pile en BP
            PUSH    SI           ; Sauvegarde index source
                                 ;
            MOV     SI,(BP+ØEH)  ; Chargement adresse variable A en SI
            MOV     AX,[SI]      ; Transfert valeur variable A en AX
            MOV     SI,(BP+ØAH)  ; Chargement adresse variable B en SI
            MOV     BX,[SI]      ; Transfert valeur variable B en BX
            ADD     AX,BX        ; Addition de A et de B (résultat dans AX)
            MOV     SI,(BP+Ø6H)  ; Chargement adresse variable C en SI
            MOV     [SI],AX      ; Transfert contenu de AX à la variable A
                                 ;
            POP     SI           ; Rétablissement index source
            POP     BP           ; Rétablissement pointeur de base
                                 ;
ADDITION    ENDP                 ; Fin de la procédure d'addition
CODE        ENDS                 ; Fin du segment de code
            END                  ; Fin de la routine
```

Après déclaration et affectation du segment de code (c'est lui qui renferme le code objet de la routine...), la routine d'addition est déclarée en procédure de type FAR.

Ceci étant, les lignes suivantes sauvegardent l'index source SI (on eut pu tout aussi bien employer l'index destinataire DI...) et le pointeur de base BP avant que ne soit attribué à ce dernier la valeur du pointeur de pile SP. Or, c'est cette valeur qui, mise dans BP, va nous servir à gérer les paramètres dans la pile.

Vous noterez que l'on ne transmet pas à la routine les valeurs numériques des paramètres, mais bien au contraire l'adresse où l'on est susceptible de les trouver et que ceux-ci s'empilent dans l'ordre inverse de leur écriture, le dernier paramètre transmis étant affecté du décalage minimum de 06. Ceci explique pourquoi le transfert des paramètres dans les registres AX et BX se passe en deux temps, l'index source recevant l'adresse dans un premier temps, adresse qui sert ensuite à transférer la valeur des paramètres d'entrée dans les registres AX et BX dans un second temps.

Les adresses étant transmises sous la forme segment + décalage, il s'en suit que le pas de décalage employé au sein de la routine pour lire la pile est de 4 octets, que les paramètres lui soient transmis (et donc les variables employées...) sous la forme d'entiers sur deux octets ou de type simple ou double précision. Ne tombez pas dans le piège qui consiste à croire que le pas de décalage est dépendant du type de variable choisie : il n'en est rien !...

L'addition étant effectuée et le résultat expédié vers la variable C, il ne reste plus qu'à rétablir dans leur valeur d'origine l'index source et le pointeur de base avant de rendre la main au programme d'appel. Notez au passage l'absence de RET final (non, ce n'est pas un oubli...). Le Turbo Basic est ainsi fait qu'il gère lui-même la pile au retour de la routine, ce qui oblige le programmeur à modifier quelque peu ses habitudes...

Procédure d'intégration au Turbo Basic

Ainsi, lorsque la longueur de la routine à intégrer est considérable et ne peut raisonnablement être écrite sous forme de codes-opération, le Turbo Basic vous propose de l'inclure au programme en tant que fichier .COM au travers d'un sous-programme de type INLINE.

Pour ce faire, il vous faut commencer par assembler le code source de la routine à l'aide d'un Macro-Assembleur (en l'occurence il s'agit de celui de Microsoft...) de façon à obtenir un fichier .OBJ :

```
>MASM ADDITION;
```

A partir de ce fichier objet, vous allez produire un fichier exécutable d'extension .EXE en lançant comme suit l'édition de liens :

```
>LINK ADDITION;
```

Ne vous préoccupez pas du message d'avertissement « Warning : no stack segment » que va vous délivrer l'éditeur de liens. Cela tient, comme vous l'aviez deviné, à l'absence de segment de pile au sein de la routine. Or comme ce segment n'est pas indispensable...

Ceci étant, vous allez maintenant convertir ce fichier exécutable en un fichier binaire d'extension .COM en lançant comme suit l'utilitaire de conversion EXE2BIN livrée avec le MS DOS de la machine :

```
>EXE2BIN ADDITION ADDITION.COM
```

Notez la répétition du nom du fichier, suivi pour le second de l'extension .COM à la suite du nom de l'utilitaire. Sans cette double indication, celui-ci vous délivrerait un fichier binaire d'extension .BIN qu'il vous faudrait ensuite renommer en .COM à l'aide de la commande RENAME.

Programme d'essai en Turbo Basic

La mise en forme du fichier binaire d'extension .COM de la routine étant chose faite, il ne vous reste plus qu'à le charger au sein du programme par le biais d'un sous-programme de type INLINE, comme dans l'exemple que voici :

```
DEFINT A-Z
A=5:B=7:C=Ø
CALL Addition(A,B,C)
PRINT "5+7=";C
END

SUB Addition INLINE
    $INLINE "ADDITION.COM"
END SUB
```

Comparé au programme précédent, celui-ci n'en diffère que par la présence d'une seule instruction $INLINE dans le sous-programme de type INLINE. Or, comme vous pouvez le constater, cette ligne unique assure le chargement du fichier binaire d'extension .COM de la routine d'addition.

Quant au reste du programme, il est on ne peut plus dépouillé. Après avoir déclaré les variables comme étant entières et leur avoir attribué une valeur, la routine est appelée par un CALL, comme s'il s'agissait d'un quelconque sous-programme. Simple et efficace à la fois... !

Cela étant, il ne vous reste plus qu'à exécuter ce petit programme d'essai afin de vous assurer que cette procédure d'intégration des fichiers .COM fonctionne correctement.

Pour le reste, à savoir la production d'un module exécutable autonome du programme Basic intégrant une (ou plusieurs?...) routines, il suffit de choisir l'option « Compile to EXE file » dans le menu « Options » pour que, lors de la compilation, le TurboBasic dépose sur la disquette ou le disque dur un module exécutable autonome.

Récapitulatif de la procédure d'appel

Turbo Basic version 1.0 Fichiers .COM	
Instruction d'appel	CALL
Paramètres	variables
Transmis sous forme	d'adresses
Empilage dans l'ordre	inverse d'écriture
Procédure de type	FAR
Décalage initial	06H
Pas de décalage	4 octets
Fin de routine	pas de RET
Directives à spécifier	aucune

PASCAL MICROSOFT
Procédure externe

Bien qu'un peu ancien, le langage Pascal possède encore des adeptes et mérite donc que l'on s'y attarde quelque peu. L'intégration de routines s'y opère relativement facilement au travers de procédures ou de fonctions que l'on déclare comme étant externes au corps du programme et que l'on intègre à celui-ci au moment de l'édition de liens.

La procédure d'appel propre au Pascal, ce que la littérature anglo-saxonne nomme « the Pascal Calling Procedure » ressemble à quelques détails près à celle employée par les différents Basics, à savoir que les paramètres peuvent être passés sous la forme d'adresses ou de valeurs, se gèrent sous 2, 4 ou 8 octets selon leur nature et s'entassent dans la pile dans l'ordre inverse de leur écriture.

Oui, vous avez bien lu, les paramètres peuvent être passés sous la forme d'adresses ou de valeurs :
- *Passés sous forme d'adresses,* les paramètres seront obligatoirement des variables (comme en Basic).
- *Passés sous forme de valeurs,* ils pourront être indifféremment des variables et/ou des valeurs numériques.

Fort de ces précisions, nous allons examiner 2 exemples d'appel de routines en tant que procédures externes, le premier consistant à passer les paramètres sous forme d'adresses et le second sous forme de valeurs.

Passage de paramètres sous forme d'adresses

```
;          ************************************************
;          Routine d'addition adaptée au Pascal Microsoft
;             (procédure externe - passage d'adresses)
;          ************************************************

CODE       SEGMENT BYTE 'CODE'  ; Déclaration du segment de code.
           PUBLIC  ADDITION     ; Routine de type "public".
           ASSUME  CS:CODE      ; Affectation du segment de code.
ADDITION   PROC    FAR          ; Déclaration de procédure FAR.
                                ;
           PUSH    BP           ; Sauvegarde pointeur de base.
           MOV     BP,SP        ; Transfert pointeur de pile en BP.
           PUSH    SI           ; Sauvegarde index source.
                                ;
           MOV     SI,(BP+ØAH)  ; Chargement adresse de A en SI.
           MOV     AX,[SI]      ; Transfert valeur de A en AX.
           MOV     SI,(BP+Ø8H)  ; Chargement adresse de B en SI.
           MOV     BX,[SI]      ; Transfert valeur de B en BX.
           ADD     AX,BX        ; Addition de A + B (résultat en AX).
           MOV     SI,(BP+Ø6H)  ; Chargement adresse de C en SI.
           MOV     [SI],AX      ; Transfert contenu de AX à la variable C.
                                ;
           POP     SI           ; Rétablissement index source.
           POP     BP           ; Rétablissement pointeur de base.
           RET     6            ; ... et retour à l'envoyeur.
                                ;
ADDITION   ENDP                 ; Fin de la procédure d'addition.
           CODE    ENDS         ; Fin du segment de code.
           END                  ; Fin de la routine.
```

Après déclaration et affectation du segment de code (c'est lui qui renferme le code objet de la routine...), la routine d'addition est déclarée en procédure de type FAR.

Ceci étant, les lignes suivantes sauvegardent l'index source SI (on eut pu tout aussi bien employer l'index destinataire DI...) et le pointeur de base BP avant que ne soit attribué à ce dernier la valeur du pointeur de pile SP. Or, c'est cette valeur qui, mise dans BP, va nous servir à gérer les paramètres dans la pile.

Vous noterez que l'on transmet à la routine l'adresse où l'on est susceptible de trouver les paramètres et que ceux-ci s'empilent dans l'ordre inverse de leur écriture, le dernier paramètre transmis étant affecté du décalage minimum de 06. Ceci explique pourquoi le transfert des paramètres dans les registres AX et BX se passe en deux temps, l'index source recevant l'adresse dans un premier temps, adresse qui sert ensuite à transférer la valeur des paramètres d'entrée dans les registres AX et BX dans un second temps.

Dans ces conditions, le pas de décalage employé au sein de la routine pour lire la pile est de 2 octets, que les paramètres lui soient transmis (et donc les variables employées...) sous la forme d'entiers sur deux octets ou de type simple ou double précision. Ne tombez pas dans le piège qui consiste à croire que le pas de décalage est dépendant du type de variable choisie : il n'en est rien !...

L'addition étant effectuée et le résultat expédié vers la variable C, il ne reste plus qu'à rétablir dans leur valeur d'origine l'index source et le pointeur de base avant de rendre la main au programme d'appel. Notez au passage que le RET final est suivi du nombre 6, ce qui équivaut à 3 paramètres de 2 octets chacun et assure la gestion correcte de la pile.

Remarquez également que la routine doit être déclarée de type PUBLIC afin de pouvoir être reconnue par l'éditeur de liens. Dans le même ordre d'idées, il vous faut rajouter l'indication 'CODE' à la suite de la déclaration du segment de code de façon à ce que l'éditeur de liens sache où le situer.

Important : *La routine écrite sous cette forme ressemble à s'y méprendre à celle appelée par les Basics (même paramètres transmis sous forme d'adresses, même position des paramètres dans la pile, même RET final tenant compte du nombre d'octets des paramètres, même déclaration de type PUBLIC de la routine et de type 'CODE' pour le code objet, etc...). Ce point est à noter car cette procédure d'appel permet d'utiliser sous Pascal des routines écrites pour le Basic.*

Programme d'essai en Pascal

Examinons le petit programme d'essai que voici qui déclare la routine en tant que procédure externe et qui assure la transmission de variables vers, et en retour de celle-ci :

```
program ESSAI1(input,output);
var a,b,c:integer;
procedure addition(var a,b,c:integer);extern;

begin
    a:=5;
    b:=7;
    addition(a,b,c);
    write('5+7=',c:1);
end.
```

Comme vous pouvez le constater par vous-même, le programme appelle la procédure d'addition après que celle-ci ait été déclarée comme étant externe et passant 3 variables entières (a, b, c) à la routine. Pour le reste, il s'agit d'affecter les valeurs 5 et 7 aux variables A et B et l'addition effectuée, à afficher le résultat contenu dans C.

Procédure d'intégration au Pascal

La routine et le programme étant écrits, il vous faut maintenant vérifier si l'intégration de l'un à l'autre s'effectue correctement lors de l'édition de liens. Pour ce faire, vous allez devoir taper les ordres suivants :

```
>MASM ADDITION;

>PAS1 ESSAI;

>PAS2

>LINK ESSAI ADDITION;
```

Pour mémoire, ces ordres correspondent respectivement à l'assemblage du code objet de la routine d'addition, suivi des 2 passes du compilateur Pascal (la 3[e] n'étant pas nécessaire dans ce cas...), suivie à son tour par l'édition de liens au cours de laquelle s'effectue l'intégration du fichier .OBJ de la routine au fichier exécutable.

A ce propos, notez qu'il est inutile de transformer le fichier .OBJ de la routine en un fichier .LIB, pourvu que l'on rassemble les commandes d'édition de liens sur une seule ligne.

Ceci fait, il reste à essayer le fichier ESSAI.EXE résultant. Normalement, vous devriez aboutir à quelque chose ressemblant à ceci :

```
>ESSAI
5+7=12
>
```

Récapitulatif de la procédure d'appel

Pascal Microsoft Procédure externe		
Instruction d'appel		aucune
Paramètres		variables
Transmis sous forme		d'adresses
Empilage dans l'ordre		inverse d'écriture
Procédure de type		FAR
Décalage initial		06H
Pas de décalage		2 octets
Fin de routine		RET n(*)
Directives à spécifier	:	PUBLIC et 'CODE'

(*) n = Nombre de paramètres transmis X pas de décalage.

Passage de paramètres sous forme de valeurs

Ce premier point étant acquis, nous allons maintenant examiner le processus permettant de passer à la routine, non plus les adresses des paramètres, mais leur valeur.

Pour cela, il vous faut réécrire comme suit la routine d'addition :

```
;          ********************************************
;          Routine d'addition adaptée au Pascal Microsoft
;             (procédure externe - passage de valeurs)
;          ********************************************

CODE       SEGMENT BYTE 'CODE'  ; Déclaration du segment de code.
           PUBLIC  ADDITION     ; Routine de type "public".
           ASSUME  CS:CODE      ; Affectation du segment de code.
ADDITION   PROC    FAR          ; Déclaration de procédure FAR.
                                ;
           PUSH    BP           ; Sauvegarde pointeur de base.
           MOV     BP,SP        ; Transfert pointeur de pile en BP.
           PUSH    SI           ; Sauvegarde index source.
                                ;
           MOV     AX,(BP+ØAH)  ; Transfert premier paramètre en AX.
           MOV     BX,(BP+Ø8H)  ; Transfert deuxième paramètre en BX.
           ADD     AX,BX        ; Addition de A avec B (résultat en AX).
           MOV     SI,(BP+Ø6H)  ; Chargement adresse variable C en SI.
           MOV     [SI],AX      ; Transfert contenu de AX à la variable C.
                                ;
           POP     SI           ; Rétablissement index source.
           POP     BP           ; Rétablissement pointeur de base.
           RET     6            ; ... et retour à l'envoyeur.
                                ;
ADDITION   ENDP                 ; Fin de la procédure d'addition.
           CODE    ENDS         ; Fin du segment de code.
           END                  ; Fin de la routine.
```

Quelles différences constatez-vous entre cette version de la routine et la précédente ?... Que 2 paramètres sur 3 sont transmis sous forme de valeurs (fini l'index SI qui reçoit l'adresse...), le 3eme se devant, quant à lui, d'être transmis sous forme d'adresse, puisqu'il s'agit du résultat de l'addition qui doit remonter vers le programme appelant.

Programme d'essai en Pascal

Au tour du programme appelant à être modifié :

```
program ESSAI2(input,output);
var b,c:integer;

procedure addition(a,b:integer;var c:integer);extern;

begin
    b:=7;
    addition(5,b,c);
    write('5+7=',c:1);
end.
```

Les différences entre cette version et la précédente ?... Tout d'abord, la déclaration de la procédure externe spécifie que les 2 premiers paramètres sont transmis sous forme de valeurs même si, dans notre cas, seul le 1[er] paramètre l'est effectivement.

Pourquoi cela ?... Parce que cette forme d'écriture autorise indifféremment le passage de variables ou de valeurs numériques à la routine, la preuve en est le mélange des deux formes auquel nous avons procédé (le 1[er] paramètre étant une valeur numérique et le 2[nd] une variable...).

Bien entendu, le 3[eme] paramètre est et doit demeurer une variable, car il correspond à la remontée du résultat de l'addition provenant de la routine.

Procédure d'intégration au Pascal

La routine et le programme étant écrits, il vous reste à vérifier si l'intégration de l'un à l'autre s'effectue correctement lors de l'édition de liens. Pour ce faire. vous allez devoir taper les ordres suivants :

```
>MASM ADDITION;

>PAS1 ESSAI;

>PAS2

>LINK ESSAI ADDITION;
```

Pour mémoire, ces ordres correspondent respectivement à l'assemblage du code objet de la routine d'addition, suivi des 2 passes du compilateur Pascal (la 3eme n'étant pas nécessaire dans ce cas...), suivie à son tour par l'édition de liens au cours de laquelle s'effectue l'intégration du fichier .OBJ de la routine au fichier exécutable.

A ce propos, notez qu'il est inutile de transformer le fichier .OBJ de la routine en un fichier .LIB, pourvu que l'on rassemble les commandes d'édition de liens sur une seule ligne.

Ceci fait, il reste à essayer le fichier ESSAI.EXE résultant. Normalement, vous devriez aboutir à quelque chose ressemblant à ceci :

```
>ESSAI
5+7=12
>
```

Récapitulatif de la procédure d'appel

Pascal Microsoft Procédure externe		
Instruction d'appel	:	aucune
Paramètres		variables et valeurs numériques
Transmis sous forme		de valeurs
Empilage dans l'ordre	:	inverse d'écriture
Procédure de type	:	FAR
Décalage initial	:	06H
Pas de décalage	:	(2 à 8 octets pour les nombres)
Fin de routine		RET n(*)
Directives à spécifier	:	PUBLIC et 'CODE'

(*) n = Nombre total d'octets correspondant aux paramètres.

PASCAL MICROSOFT

Fonction externe

L'assimilation d'une routine Assembleur à une fonction externe est une des possibilités offertes par le Pascal Microsoft. Sa mise en œuvre est à conseiller chaque fois qu'une routine doit réexpédier un résultat vers le programme appelant, la procédure d'appel propre à une fonction se chargeant automatiquement de cette « basse besogne ».

La procédure d'appel propre au Pascal, ce que la littérature anglo-saxonne nomme « the Pascal Calling Procedure » ressemble à quelques détails près à celle employée par les différents Basics, à savoir que les paramètres peuvent être passés sous la forme d'adresses ou de valeurs, se gèrent sous 2, 4 ou 8 octets selon leur nature et s'entassent dans la pile dans l'ordre inverse de leur écriture.

Oui, vous avez bien lu, les paramètres peuvent être passés sous la forme d'adresses ou de valeurs :

- *Passés sous forme d'adresses*, les paramètres seront obligatoirement des variables (comme en Basic).
- *Passés sous forme de valeurs*, ils pourront être indifféremment des variables et/ou des valeurs numériques.

Bien que les deux formes soient possibles, nous n'envisageons de passer les paramètres à la routine que sous forme de valeurs dans l'exemple qui suit.

Routine en Assembleur

```
;           ****************************************************
;           Routine d'addition adaptée au Pascal Microsoft
;              (fonction externe - passage de valeurs)
;           ****************************************************

CODE        SEGMENT BYTE 'CODE'   ; Déclaration du segment de code.
            PUBLIC  ADDITION      ; Routine de type "public".
            ASSUME  CS:CODE       ; Affectation du segment de code.
ADDITION    PROC    FAR           ; Déclaration de procédure FAR.
                                  ;
            PUSH    BP            ; Sauvegarde pointeur de base.
            MOV     BP,SP         ; Transfert pointeur de pile en BP.
                                  ;
            MOV     AX,(BP+Ø8H)   ; Transfert 1er paramètre en AX.
            MOV     BX,(BP+Ø6H)   ; Transfert 2nd paramètre en BX.
            ADD     AX,BX         ; Addition de A + B (résultat en AX).
                                  ;
            POP     BP            ; Rétablissement pointeur de base.
            RET     4             ; ... et retour à l'envoyeur.
                                  ;
ADDITION    ENDP                  ; Fin de la procédure d'addition.
            CODE    ENDS          ; Fin du segment de code.
            END                   ; Fin de la routine.
```

Qu'est-ce qui a changé dans l'écriture de cette routine ?... Tout d'abord, la disparition des 2 lignes de renvoi du résultat de l'addition contenu dans AX, celui-ci étant désormais pris en charge automatiquement par la procédure d'appel de la fonction.

Ensuite, ce ne sont plus 3 mais 2 paramètres que l'on transmet à la routine d'addition. Dans ces conditions, les décalages de lecture de la pile sont à diminuer de 2 (08H pour le 1er paramètre et 06H pour le second au lieu de 0AH et 08H...), ainsi que le nombre d'octets figurant à la suite du RET final qui passent de 6 à 4.

Enfin, il n'y a plus lieu de sauvegarder et de rétablir l'index source SI, puisqu'il n'est plus d'aucune utilité dans le passage des paramètres. Pour mémoire, l'index source recevait dans un premier temps l'adresse de la variable, adresse qui servait ensuite à transférer la valeur de cette variable dans un registre. Or, comme on ne transmet plus d'adresse...

Programme d'essai en Pascal

Examinons le petit programme d'essai que voici qui déclare la routine en tant que fonction externe et qui assure le passage des paramètres A et B de l'addition :

```
program ESSAI3(input,output);
var b,c:integer;

function addition(a,b:integer):integer;extern;

begin
   b:=7;
   c:=addition(5,b);
   write('5+7=',c:1);
end.
```

Que pouvons-nous constater ?... Que la déclaration d'une fonction externe ressemble à s'y méprendre à celle d'une procédure, la seule différence consistant à préciser, à l'extérieur des parenthèses, la nature de la valeur numérique renvoyée par la fonction.

Cela étant, il suffit d'attribuer cette valeur à une variable (c en l'occurence) pour disposer du résultat de l'addition.

De surcroît, vous noterez que le passage des paramètres sous forme de valeurs autorise ces derniers à être indifféremment des variables ou des valeurs numériques, la preuve en est le mélange des deux formes auquel nous nous sommes livrés (le 1er paramètre étant une valeur numérique et le 2nd une variable...).

Procédure d'intégration au Pascal

La routine et le programme étant écrits, il vous reste à vérifier si l'intégration de l'un à l'autre s'effectue correctement lors de l'édition de liens. Pour ce faire, vous allez devoir taper les ordres suivants :

```
>MASM ADDITION;

>PAS1 ESSAI;

>PAS2

>LINK ESSAI ADDITION;
```

Pour mémoire, ces ordres correspondent respectivement à l'assemblage du code objet de la routine d'addition, suivi des 2 passes du compilateur Pascal (la 3eme n'étant pas nécessaire dans ce cas...), suivie à son tour par l'édition de liens au cours de laquelle s'effectue l'intégration du fichier .OBJ de la routine au fichier exécutable.

A ce propos, notez qu'il est inutile de transformer le fichier .OBJ de la routine en un fichier .LIB, pourvu que l'on rassemble les commandes d'édition de liens sur une seule ligne.

Ceci fait, il reste à essayer le fichier ESSAI.EXE résultant. Normalement, vous devriez aboutir à quelque chose ressemblant à ceci :

```
>ESSAI
5+7=12
>
```

Récapitulatif de la procédure d'appel

Pascal Microsoft Fonction externe		
Instruction d'appel	:	aucune
Paramètres		variables et valeurs numériques (1)
Transmis sous forme		de valeurs (1)
Empilage dans l'ordre		inverse d'écriture
Procédure de type		FAR
Décalage initial		06H
Pas de décalage		(2 à 8 octets pour les nombres) (1)
Fin de routine		RET n(*)
Directives à spécifier	:	PUBLIC et 'CODE'

(*) n = Nombre total d'octets correspondant aux paramètres.

(1) Il est à noter que la procédure indiquée correspond au passage de paramètres sous la forme de valeurs. Dans le cas où ceux-ci seraient transmis sous forme d'adresses, il ne serait possible de passer que des variables, le pas de décalage étant alors de 2 octets.

TURBO PASCAL version 4.0

Code en ligne

Possédant à la fois les caractéristiques du Pascal et quelques-unes des innovations propres, à Borland, le Turbo Pascal dans sa version 4.0 permet d'intégrer les routines écrites en Assembleur par le biais de procédures et de fonctions de type externe, comme il peut intégrer du code en ligne au travers de directives inline.

La procédure d'appel propre au Turbo Pascal, est identique à quelques détails près à celle du Pascal Microsoft, à savoir que les paramètres peuvent être indifféremment passés à la routine sous forme d'adresses ou de valeurs, et que, selon le cas, ils sont gérés sur 2, 4 ou 8 octets et s'entassent dans la pile dans l'ordre inverse de leur écriture.

Oui, vous avez bien lu, les paramètres peuvent être passés sous forme d'adresses ou de valeurs :

- *sous forme d'adresses*, ils sont gérés sur 4 octets (et non 2 comme en Pascal) et ne peuvent accepter que des variables.
- *sous forme de valeurs*, ils sont gérés sur autant d'octets qu'il est nécessaire et peuvent accepter indifféremment variables et valeurs numériques.

Bien que les deux formes soient possibles, nous n'envisageons de passer les paramètres à la routine que sous la forme de valeurs dans l'exemple qui suit. Cette façon de procéder présente à nos yeux un double avantage, celui d'accepter les paramètres sous forme de variables et de valeurs numériques et d'harmoniser les programmes et routines du Turbo Pascal avec le Pascal (le passage d'adresses s'effectuant sur 4 octets en Turbo Pascal et sur 2 octets en Pascal...).

Routine en Assembleur

L'incorporation des codes-opération de la routine à la suite d'une directive inline nécessite au préalable que soient connus ces fameux codes.

Pour ce faire, il faut écrire la routine en Assembleur et, après diverses opérations que nous allons décrire, la transformer en un fichier binaire d'extension .BIN que l'on désassemble sous DEBUG.

Afin d'illustrer ce qui va suivre, nous avons choisi une routine d'addition de 2 nombres entiers qui, bien que totalement inutile (le Turbo Pascal en est déjà pourvu...), va nous permettre de contrôler la transmission correcte des paramètres entre le programme et la routine.

```
;          ************************************************
;          Routine d'addition adaptée au TurboPascal 4.Ø
;                            Code en ligne
;          ************************************************

CODE       SEGMENT BYTE          ; Déclaration du segment de code.
           ASSUME  CS:CODE       ; Affectation du segment de code.
ADDITION   PROC    FAR           ; Déclaration de procédure FAR.
                                 ;
           POP     BX            ; Récupération du 2nd paramètre.
           POP     AX            ; Récupération du 1er paramètre.
           ADD     AX,BX         ; Addition de A + B (résultat en AX).
                                 ;
ADDITION   ENDP                  ; Fin de la procédure d'addition.
           CODE    ENDS          ; Fin du segment de code.
           END                   ; Fin de la routine.
```

Cette routine amène de nombreux commentaires. Tout d'abord, il vous faut savoir qu'elle va être considérée comme une fonction interne, ce qui évite d'avoir à prévoir le retour du résultat de l'addition vers le programme Turbo Pascal, la procédure d'appel de la fonction interne y pourvoyant à notre place.

Ensuite, vous n'avez pas été sans remarquer l'emploi de deux instructions POP pour transfèrer les valeurs numériques des paramètres du dessus de la pile vers les registres. Cela tient aux directives inline que le programme considère comme une macro-instruction, interdissant ainsi l'emploi des habituels codes d'entrée et de sortie des paramètres ainsi que du RET final.

Remarquez également l'ordre de prélèvement des paramètres sur le dessus de la pile. C'est tout le contraire de l'habituelle procédure d'appel du Pascal (à qui se fier...) puisque le paramètre figurant sur le haut de la pile est le dernier à avoir été transmis.

La routine tapée, il vous faut assembler son code source à l'aide d'un Macro-Assembleur (en l'occurence il s'agit de celui de Microsoft...) de façon à obtenir un fichier .OBJ :

```
>MASM ADDITION;
```

A partir de ce fichier objet, vous allez produire un fichier exécutable d'extension .EXE en lançant comme suit l'édition de liens (l'éditeur de liens à employer étant celui livré avec le Macro-Assembleur) :

```
>LINK ADDITION;
```

Ne vous préoccupez pas du message d'avertissement « Warning : no stack segment » que va vous délivrer l'éditeur de liens. Cela tient, comme vous l'aviez deviné, à l'absence de segment de pile au sein de la routine. Or comme ce segment n'est pas indispensable...

Ceci étant, vous allez maintenant convertir ce fichier exécutable en un fichier binaire d'extension .BIN en lançant comme suit l'utilitaire de conversion EXE2BIN livrée avec le MS DOS de la machine :

```
>EXE2BIN ADDITION
```

Vous obtenez alors un fichier ADDITION.BIN de 4 octets de long qu'il vous suffit de désassembler avec l'utilitaire DEBUG du MS DOS pour obtenir les codes-opération de la routine :

```
DEBUG ADDITION.BIN
-D
52E8:Ø1ØØ 5B            POP     BX
52E8:Ø1Ø1 58            POP     AX
52E8:Ø1Ø2 Ø3C3          ADD     AX,BX

              etc... (la suite étant sans intérêt)

-Q
```

Programme d'essai en Turbo Pascal

Fort des valeurs des codes-opération de la routine d'addition (version revue et corrigée pour le code en ligne...), il ne nous reste plus qu'à les incorporer au sein du programme en Turbo Pascal à la suite d'une directive inline, comme dans le programme ci-après :

```
program ESSAI;
var b,c:integer;

function Addition(a,b:integer):integer;
inline(
   $5B/          { POP    BX       }
   $58/          { POP    AX       }
   $Ø3/$C3);     { ADD    AX,BX    }

begin
   b:=7;
   c:=Addition(5,b);
   write('5+7=',c);
end.
```

Comme prévu, vous retrouvez les 4 codes-opération de la routine d'addition au sein de la fonction Addition. Notez au passage l'écriture qui signale le passage des paramètres sous forme de valeurs entières, le retour de la fonction étant également un entier.

De surcroît, vous noterez que le passage des paramètres sous forme de valeurs autorise ceux-ci à être indifféremment des variables ou des valeurs numériques, la preuve en est le mélange des deux auquel nous nous sommes livrés (le 1er paramètre étant une variable numérique et le 2nd une variable...).

Cela étant, il ne vous reste plus qu'à exécuter ce programme d'essai afin de vous assurer que cette procédure d'intégration dite du « code en ligne » fonctionne correctement.

Pour le reste, à savoir la production d'un module exécutable autonome du programme Turbo Pascal intégrant une (ou plusieurs...) routines, il suffit de choisir l'option « Compile, Make ou Build dans le menu « Compile » et préciser la destination « Disk » pour que, lors de la compilation, le compilateur et l'éditeur de liens du Turbo Pascal dépose sur la disquette ou le disque dur un module exécutable autonome.

Récapitulatif de la procédure d'appel

Turbo Pascal version 4.0 Code en ligne	
Instruction d'appel	aucune
Paramètres	variables et valeurs numériques
Transmis sous forme	de valeurs
Empilage dans l'ordre	normal d'écriture
Procédure de type :	inutile
Décalage initial :	inutile
Pas de décalage :	inutile
Fin de routine :	inutile
Directives à spécifier	aucune

TURBO PASCAL version 4.0
Procédure externe

Parmi les nombreuses possibilités du Turbo Pascal 4.0, il en est une qui consiste à assimiler une routine écrite en Assembleur à une procédure externe. Simple et pratique d'emploi, cette approche est à conseiller pour les routines qui ne renvoient aucune information vers le programme appelant ou, au contraire, plusieurs.

La procédure d'appel propre au Turbo Pascal, est identique à quelques détails près à celle du Pascal Microsoft, à savoir que les paramètres peuvent être indifféremment passés à la routine sous forme d'adresses ou de valeurs et que, selon le cas, ils sont gérés sur 2, 4, 6, 8 octets ou plus et s'entassent dans la pile dans l'ordre inverse de leur écriture.

Oui, vous avez bien lu, les paramètres peuvent être passés sous forme d'adresses ou de valeurs :

- *sous forme d'adresses,* ils sont gérés sur 4 octets (et non 2 comme en Pascal) et ne peuvent accepter que des variables.
- *sous forme de valeurs,* ils sont gérés sur autant d'octets qu'il est nécessaire et peuvent accepter indifféremment variables et valeurs numériques.

Afin d'illustrer ce qui va suivre, nous avons choisi une routine d'addition de 2 nombres entiers qui, bien que totalement inutile (le Turbo Pascal en est déjà pourvu...), va nous permettre d'assimiler les subtilités inhérentes au passage d'adresses ou de valeurs à une procédure externe.

Routine en Assembleur

```
;           *********************************************
;           Routine d'addition adaptée au TurboPascal 4.Ø
;              (fonction externe - passage de valeurs)
;           *********************************************

CODE        SEGMENT BYTE          ; Déclaration du segment de code.
            ASSUME  CS:CODE       ; Affectation du segment de code.
            PUBLIC  Addition      ; Routine de type "public".
Addition    PROC    FAR           ; Déclaration de procédure FAR.
                                  ;
            PUSH    BP            ; Sauvegarde pointeur de base.
            MOV     BP,SP         ; Transfert pointeur de pile en BP.
                                  ;
            MOV     AX,(BP+Ø8H)   ; Transfert 1er paramètre en AX.
            MOV     BX,(BP+Ø6H)   ; Transfert 2nd paramètre en BX.
            ADD     AX,BX         ; Addition de A + B (résultat en AX).
                                  ;
            POP     BP            ; Rétablissement pointeur de base
            RET                   ; ... et retour à l'envoyeur.
                                  ;
Addition    ENDP                  ; Fin de la procédure d'addition.
            CODE    ENDS          ; Fin du segment de code.
            END                   ; Fin de la routine.
```

Après déclaration et affectation du segment de code (c'est lui qui renferme le code objet de la routine...), la routine d'addition est déclarée en procédure de type FAR.

Ceci étant, les lignes suivantes sauvegardent l'index source SI (on eut pu tout aussi bien employer l'index destinataire DI...) et le pointeur de base BP avant que en soit attribué à ce dernier la valeur du pointeur de pile SP. Or, c'est cette valeur qui, mise dans BP, va nous servir à gérer les paramètres dans la pile.

Vous noterez que l'on transmet les 2 premiers paramètres à la routine sous forme de valeurs, alors que le 3ème qui correspond au résultat de l'addition est transmis sous forme d'adresse, l'index source SI recevant l'adresse de la variable C dans un premier temps, adresse qui sert ensuite à transférer le contenu du registre AX vers la variable C dans un second temps.

Pourquoi cela ?... **Parce qu'une procédure externe ne peut recevoir d'informations en provenance d'une routine que si celles-ci sont passées sous forme d'adresses. Cette règle est impérative.**

Quant à l'autre sens de transmission, c'est-à-dire du programme appelant vers la routine, toute liberté est permise. C'est pourquoi nous avons choisi de passer les paramètres à la routine sous forme de valeurs, cette façon de procéder présentant à nos yeux un avantage certain, celui de pouvoir utiliser indifféremment des variables ou des valeurs numériques en guise de paramètres.

Dans ces conditions, le pas de décalage employé au sein de la routine pour lire la pile est de 2 octets pour les 2 premiers paramètres (car il s'agit d'entiers transmis sous forme de valeurs...), alors qu'il est de 4 octets pour le 3eme paramètre transmis sous forme d'adresse.

L'addition étant effectuée et le résultat expédié vers la variable C, il ne reste plus qu'à rétablir dans leur valeur d'origine l'index source et le pointeur de base avant de rendre la main au programme d'appel. Notez au passage l'absence de nombre à la suite du RET final, la procédure d'appel au Turbo Pascal se chargeant automatiquement de ce travail.

Un dernier point est à noter : la routine doit être déclarée de type PUBLIC si l'on veut qu'elle soit reconnue par l'éditeur de liens.

Programme d'essai en Turbo Pascal

Examinons le petit programme d'essai que voici qui déclare la routine en tant que procédure externe :

```
{$F+}
{$L Addition}

program ESSAI;
var b,c:integer;

procedure Addition(a,b:integer;var c:integer);external;

begin
   b:=7;
   c:=Ø;
   Addition(5,b,c);
   write('5+7=',c);
end.
```

Comme vous pouvez le constater, le programme débute par la directive $F+ qui sélectionne la procédure « FAR », suivi de la directive $L (pour librairie...) qui précise à l'éditeur de liens qu'il lui faut inclure le fichier .OBJ de la routine d'addition au fichier exécutable lors de l'édition de liens.

Cela étant, on déclare les variables b et c comme étant entières, les 2 premiers paramètres de la procédure comme étant transmis sous forme de valeurs, le 3ème sous forme d'adresse et la procédure elle-même comme étant externe.

On affecte ensuite leurs valeurs aux variables b et c et on appelle la procédure externe. Notez que le passage des 2 premiers paramètres sous forme de valeurs autorise l'emploi de valeurs numériques et de variables, la preuve en est le mélange des deux auquel nous nous sommes livrés (le 1er paramètre étant une valeur numérique et le 2nd une variable...)

Procédure d'intégration au Turbo Pascal

Reste à vérifier que tout cela fonctionne !... Pour ce faire, il vous suffit d'assembler la routine d'addition à l'aide du macro-Assembleur MASM de Microsoft en tapant :

```
>MASM ADDITION;
```

...et, après avoir chargé le programme ESSAI.PAS dans l'éditeur du TurboPascal, à l'exécuter en sélectionnant l'option RUN. Un point c'est tout !...

Pour le reste, à savoir l'obtention d'un module exécutable autonome du programme Turbo Pascal intégrant une (ou plusieurs...) routines Assembleur, il suffit de sélectionner au choix l'option « Compile, Make ou Build » dans le menu « Compile » et préciser la destination « Disk » pour que, lors de la compilation, le compilateur et l'éditeur de liens dépose sur la disquette ou le disque dur un fichier .EXE autonome.

Récapitulatif de la procédure d'appel

Turbo Pascal version 4.0 Procédure externe		
Instruction d'appel	:	aucune
Paramètres	:	variables et valeurs numériques (1)
Transmis sous forme	:	de valeurs (1)
Empilage dans l'ordre	:	inverse d'écriture
Procédure de type	:	FAR
Décalage initial	:	06H
Pas de décalage	:	(2 à 8 octets pour les nombres) (1)
Fin de routine	:	RET
Directives à spécifier	:	PUBLIC

(1) Il est à noter que la procédure indiquée correspond au passage de paramètres sous la forme de valeurs. Dans le cas où ceux-ci seraient transmis sous forme d'adresses, il ne serait possible de passer que des variables, le pas de décalage étant alors de 4 octets.

TURBO PASCAL Version 4.0

Fonction externe

La seconde possibilité offerte par le Turbo Pascal 4.0 consiste à assimiler une routine Assembleur à une fonction externe. Sa mise en œuvre est recommandée chaque fois qu'une routine doit réexpédier une information (et une seule...) vers le programme appelant, la procédure d'appel propre à une fonction externe se chargeant à votre place de la « réexpédition ».

La procédure d'appel propre au Turbo Pascal, est identique à quelques détails près à celle du Pascal Microsoft, à savoir que les paramètres peuvent être indifféremment passés à la routine sous forme d'adresses ou de valeurs et que, selon le cas, ils sont gérés sur 2, 4, 6, 8 octets ou plus et s'entassent dans la pile dans l'ordre inverse de leur écriture.

Oui, vous avez bien lu, les paramètres peuvent être passés sous forme d'adresses ou de valeurs :

- *sous forme d'adresses,* ils sont gérés sur 4 octets (et non 2 comme en Pascal) et ne peuvent accepter que des variables.
- *sous forme de valeurs,* ils sont gérés sur autant d'octets qu'il est nécessaire et peuvent accepter indifféremment variables et valeurs numériques.

Afin d'illustrer ce qui va suivre, nous avons choisi une routine d'addition de 2 nombres entiers qui, bien que totalement inutile (le Turbo Pascal en est déjà pourvu...), va nous permettre d'assimiler les subtilités inhérentes au passage de valeurs à une fonction externe.

Routine en Assembleur

```
;          *********************************************
;          Routine d'addition adaptée au TurboPascal 4.Ø
;             (fonction externe - passage de valeurs)
;          *********************************************

CODE       SEGMENT BYTE          ; Déclaration du segment de code.
           ASSUME  CS:CODE       ; Affectation du segment de code.
           PUBLIC  Addition      ; Routine de type "public".
Addition   PROC    FAR           ; Déclaration de procédure FAR.
                                 ;
           PUSH    BP            ; Sauvegarde pointeur de base.
           MOV     BP,SP         ; Transfert pointeur de pile en BP.
                                 ;
           MOV     AX,(BP+Ø8H)   ; Transfert 1er paramètre en AX.
           MOV     BX,(BP+Ø6H)   ; Transfert 2nd paramètre en BX.
           ADD     AX,BX         ; Addition de A + B (résultat en AX).
                                 ;
           POP     BP            ; Rétablissement pointeur de base
           RET                   ; ... et retour à l'envoyeur.
                                 ;
Addition   ENDP                  ; Fin de la procédure d'addition.
           CODE    ENDS          ; Fin du segment de code.
           END                   ; Fin de la routine.
```

Après déclaration et affectation du segment de code (c'est lui qui renferme le code objet de la routine...), la routine d'addition est déclarée en procédure de type FAR.

Ceci étant, les 2 lignes suivantes sauvegardent le pointeur de base BP avant que ne soit attribué à ce dernier la valeur du pointeur de pile SP. Or, c'est cette valeur qui, mise dans BP, va nous servir à gérer les paramètres dans la pile.

Vous noterez que les 2 paramètres transmis sous forme de valeurs ne nécessitent aucunement l'emploi de l'index source SI, leur valeur étant directement transférée dans le registre adéquat. Inutile donc de sauvegarder et de rétablir l'index source.

Vous noterez également l'absence des habituelles instructions de retour du résultat de l'addition, la procédure d'appel de la fonction s'en chargeant d'elle-même. Quant au pas de décalage à employer pour lire la pile, celui-ci est de 2 octets, rendant ainsi les routines écrites pour le Pascal compatibles avec le Turbo Pascal.

L'addition étant effectuée, il ne reste plus qu'à rétablir dans sa valeur d'origine le pointeur de base avant de rendre la main au programme d'appel. Notez au passage l'absence de nombre à la suite du RET final, la procédure d'appel du Turbo Pascal se chargeant automatiquement de ce calcul à votre place.

Un dernier point est à noter : la routine doit être déclarée de type PUBLIC si l'on veut qu'elle soit reconnue par l'éditeur de liens.

Programme d'essai en TurboPascal

Examinons le petit programme d'essai que voici qui déclare la routine en tant que fonction externe :

```
{$F+}
{$L Addition}

program ESSAI;
var b,c:integer;

function Addition(a,b:integer):integer;external;

begin
   b:=7;
   c:=Addition(5,b);
   write('5+7=',c);
end.
```

Comme vous pouvez le constater, le programme débute par la directive $F+ qui sélectionne la procédure « Far », suivie de la directive $L (pour librairie...) qui précise à l'éditeur de liens qu'il lui faut inclure le fichier .OBJ de la routine d'addition au fichier exécutable lors de l'édition de liens.

Cela étant, on déclare les variables b et c comme étant entières, les 2 paramètres de la fonction comme étant transmis sous forme de valeurs, le retour de la fonction comme entier et la fonction elle-même comme externe.

On affecte ensuite sa valeur à la variable b et on appelle la fonction externe. Notez que le passage des 2 paramètres sous forme de valeurs autorise l'emploi de valeurs numériques et de variables, la preuve en est le mélange des deux auquel nous nous sommes livrés (le 1er paramètre étant une valeur numérique et le 2nd une variable...).

Procédure d'intégration au Turbo Pascal

Reste à vérifier que tout cela fonctionne !... Pour ce faire, il vous suffit d'assembler la routine d'addition à l'aide du macro-Assembleur MASM de Microsoft en tapant :

```
>MASM ADDITION;
```

...et, après avoir chargé le programme ESSAI.PAS dans l'éditeur du Turbo Pascal, à l'exécuter en sélectionnant l'option RUN. Un point c'est tout !...

Pour le reste, à savoir l'obtention d'un module exécutable autonome du programme Turbo Pascal intégrant une (ou plusieurs...) routines Assembleur, il suffit de sélectionner au choix l'option « Compile, Make ou Build » dans le menu « Compile » et préciser la destination « Disk » pour que, lors de la compilation, le compilateur et l'éditeur de liens dépose sur la disquette ou le disque dur un fichier .EXE autonome.

Récapitulatif de la procédure d'appel

Turbo Pascal version 4.0 Fonction externe		
Instruction d'appel	:	aucune
Paramètres	:	variables et valeurs numériques (1)
Transmis sous forme	:	de valeurs (1)
Empilage dans l'ordre	:	inverse d'écriture
Procédure de type	:	FAR
Décalage initial	:	06H
Pas de décalage	:	(2 à 8 octets pour les nombres) (1)
Fin de routine	:	RET
Directives à spécifier	:	PUBLIC

(1) Il est à noter que la procédure indiquée correspond au passage de paramètres sous la forme de valeurs. Dans le cas où ceux-ci seraient transmis sous forme d'adresses, il ne serait possible de passer que des variables, le pas de décalage étant alors de 4 octets.

FORTRAN 77

Sous-programme

Ancêtre des langages de programmation présentés dans cet ouvrage, le Fortran 77 n'en demeure pas moins l'un des outils les plus familiers du développeur, ne serait-ce que parce qu'il fut le premier des langages qu'on lui enseignât. Capable de reconnaître une routine Assembleur comme un sous-programme ou une fonction externe, il dispose de toutes les facilités d'intégration des langages modernes.

La procédure d'appel du Fortran 77 Microsoft est identique à quelques détails près à celle du Pascal de la même marque, à savoir que les paramètres sont passés uniquement sous forme d'adresses et qu'ils peuvent être indifféremment des variables ou des valeurs numériques, qu'ils sont gérés sur 4 octets et qu'ils s'entassent dans la pile dans l'ordre inverse de leur déclaration, le premier étant placé sur le dessus de la pile et le dernier à l'adresse 06H.

Routine en Assembleur

Afin d'illustrer ce qui va suivre, nous avons choisi une routine d'addition de 2 nombres entiers qui, bien que d'une utilité discutable (le Fortran est déjà bien pourvu en la matière...), présente l'avantage de permettre le contrôle du passage correct des paramètres à la routine et vice-versa.

```
;          ****************************************
;          Routine d'addition adaptée au Fortran77
;                      Sous-programme
;          ****************************************

CODE       SEGMENT BYTE 'CODE'  ; Déclaration du segment de code.
           PUBLIC  ADD          ; Routine de type "public".
           ASSUME  CS:CODE      ; Affectation du segment de code.
ADD        PROC    FAR          ; Déclaration de procédure FAR.
                                ;
           PUSH    BP           ; Sauvegarde pointeur de base.
           MOV     BP,SP        ; Transfert pointeur de pile en BP.
           PUSH    SI           ; Sauvegarde index source.
                                ;
           MOV     SI,(BP+ØEH)  ; Chargement adresse de A en SI.
           MOV     AX,[SI]      ; Transfert valeur de A en AX.
           MOV     SI,(BP+ØAH)  ; Chargement adresse de B en SI.
           MOV     BX,[SI]      ; Transfert valeur de B en BX.
           ADD     AX,BX        ; Addition de A + B (résultat en AX).
           MOV     SI,(BP+Ø6H)  ; Chargement adresse de C en SI.
           MOV     [SI],AX      ; Retour contenu de AX à la variable C.
                                ;
           POP     SI           ; Rétablissement index source.
           POP     BP           ; Rétablissement pointeur de base.
           RET     12           ; ... et retour à l'envoyeur.
                                ;
ADD        ENDP                 ; Fin de la procédure d'addition.
CODE       ENDS                 ; Fin du segment de code.
           END                  ; Fin de la routine.
```

Après déclaration et affectation du segment de code (c'est lui qui renferme le code objet de la routine...), la routine d'addition est déclarée en procédure de type FAR. Notez à ce propos que le Fortran 77 de Microsoft n'admet que des noms de routines ne dépassant pas 6 caractères, ce qui nous a obligés à rogner de quelques caractères le nom de la routine ADDITION pour le transformer en ADD en début et en fin de déclaration de procédure.

Ceci étant, les lignes suivantes sauvegardent l'index source SI (on eut pu tout aussi bien employer l'index destinataire DI...) et le pointeur de base BP avant que ne soit attribué à ce dernier la valeur du pointeur de pile SP. Or, c'est cette valeur qui, mise dans BP, va nous servir à gérer les paramètres dans la pile.

Vous noterez que l'on ne transmet pas à la routine les valeurs numériques des paramètres, mais bien au contraire l'adresse où l'on est susceptible de les trouver et que ceux-ci s'empilent dans l'ordre inverse de leur écriture, le dernier paramètre transmis étant affecté du décalage minimum de 06. Ceci

explique pourquoi le transfert des paramètres dans les registres AX et BX se passe en deux temps, l'index source recevant l'adresse dans un premier temps, adresse qui sert ensuite à transférer la valeur des paramètres d'entrée dans les registres AX et BX dans un second temps.

Dans ces conditions, le pas de décalage employé au sein de la routine pour lire la pile est de 4 octets, que les paramètres lui soient transmis sous la forme d'entiers sur deux octets ou de type simple ou double précision. Ne tombez pas dans le piège qui consiste à croire que le pas de décalage est dépendant du type, de variable choisie : il n'en est rien !...

L'addition étant effectuée et le résultat expédié vers la variable C, il ne reste plus qu'à rétablir dans leur valeur d'origine l'index source et le pointeur de base avant de rendre la main au programme d'appel. Notez au passage que le RET final est suivi du nombre 12, ce qui équivaut à 3 paramètres de 4 octets chacun et assure la gestion correcte de la pile.

Remarquez également que la routine doit être déclarée de type PUBLIC afin de pouvoir être reconnue par l'éditeur de liens. Dans le même ordre d'idées, il vous faut rajouter l'indication 'CODE' à la suite de la déclaration du segment de code, sans quoi l'éditeur de liens ne saurait où le situer.

Important : *Une routine écrite selon la procédure d'appel du FORTRAN 77 fonctionne parfaitement en TurboPascal 4.0, à condition de la considérer comme une procédure externe et de prévoir, dans la déclaration de cette procédure, un passage des paramètres sous forme d'adresses. Seule différence sans conséquence : le RET final.*

Programme d'essai en Fortran 77

Le Fortran assure l'intégration des routines écrites en Assembleur par le biais de sous-programmes déclarés externes par défaut, ce qui revient à dire qu'il n'y a rien à préciser à leur sujet lors de l'appel d'une routine. La preuve en est le petit programme d'essai que voici :

```
      PROGRAM ESSAI1
      INTEGER B,C
      B=7
      CALL ADD(5,B,C)
      WRITE (*,100) C
100   FORMAT (1X,'5+7=',I2)
      END
```

Que dire à propos de ce programme, si ce n'est que la procédure d'appel du Fortran autorise indifféremment le passage de valeurs numériques (le nombre 5 dans cet exemple...) et de variables (B et C en l'occurence...) à la routine d'addition sans déclaration préalable.

Procédure d'intégration au Fortran 77

Cela étant, il vous faut maintenant vérifier si cette méthode d'intégration de routines fonctionne correctement. Pour ce faire, vous allez taper la suite d'ordres que voici :

```
>MASM ADD;

>FOR1 ESSAI;

>PAS2

>LINK ESSAI ADD;
```

Pour mémoire, ces ordres correspondent respectivement à l'assemblage du code objet de la routine, aux 2 passes de compilation du Fortran et, pour finir à l'édition de liens par laquelle on intègre le fichier .OBJ de la routine au code exécutable du programme.

A ce propos, notez qu'il est inutile de transformer le fichier .OBJ de la routine en une librairie d'extension .LIB, pourvu que l'on prenne la précaution de lancer l'édition de liens comme indiqué, c'est-à-dire en plaçant l'ensemble des commandes sur une même ligne.

Ceci fait, il vous reste à essayer le fichier ESSAI.EXE résultant. Normalement, vous devriez aboutir à quelque chose ressemblant à ceci :

```
>ESSAI
5+7=12
>
```

Récapitulatif de la procédure d'appel

Fortran 77 Sous-programme		
Instruction d'appel	:	CALL
Paramètres		variables et valeurs numériques
Transmis sous forme		d'adresses
Empilage dans l'ordre		inverse d'écriture
Procédure de type		FAR
Décalage initial		06H
Pas de décalage		4 octets
Fin de routine		RET n(*)
Directives à spécifier	:	PUBLIC et 'CODE'

(*) n = Nombre de paramètres transmis × pas de décalage.

FORTRAN 77

Fonction

Le Fortran 77, au même titre que les autres langages, propose de reconnaître une routine Assembleur en tant que fonction externe. Cette possibilité est à mettre en œuvre dès lors qu'une routine doit renvoyer une information (et une seule...) vers le programme appelant, la procédure d'appel inhérente à une fonction se chargeant de ce retour automatique.

La procédure d'appel du Fortran 77 Microsoft est identique à quelques détails près à celle du Pascal de la même marque, à savoir que les paramètres sont passés uniquement sous forme d'adresses et qu'ils peuvent être indifféremment des variables ou des valeurs numériques, qu'ils sont gérés sur 4 octets et qu'ils s'entassent dans la pile dans l'ordre inverse de leur déclaration, le premier étant placé sur le dessus de la pile et le dernier à l'adresse 06H.

Routine en Assembleur

Afin d'illustrer ce qui va suivre, nous avons choisi une routine d'addition de 2 nombres entiers qui, bien que d'une utilité discutable (le Fortran est déjà bien pourvu en la matière...), présente l'avantage de permettre le contrôle du passage correct des paramètres à la routine et vice-versa.

```
;          ***************************************
;          Routine d'addition adaptée au Fortran77
;                         Fonction
;          ***************************************

CODE       SEGMENT BYTE 'CODE'  ; Déclaration du segment de code.
           PUBLIC  ADD          ; Routine de type "public".
           ASSUME  CS:CODE      ; Affectation du segment de code.
ADD        PROC    FAR          ; Déclaration de procédure FAR.
                                ;
           PUSH    BP           ; Sauvegarde pointeur de base.
           MOV     BP,SP        ; Transfert pointeur de pile en BP.
           PUSH    SI           ; Sauvegarde index source.
                                ;
           MOV     SI,(BP+0AH)  ; Chargement adresse de A en SI.
           MOV     AX,[SI]      ; Transfert valeur de A en AX.
           MOV     SI,(BP+06H)  ; Chargement adresse de B en SI.
           MOV     BX,[SI]      ; Transfert valeur de B en BX.
           ADD     AX,BX        ; Addition de A + B (résultat en AX).
                                ;
           POP     SI           ; Rétablissement index source.
           POP     BP           ; Rétablissement pointeur de base.
           RET     8            ; ... et retour à l'envoyeur.
                                ;
ADD        ENDP                 ; Fin de la procédure d'addition.
CODE       ENDS                 ; Fin du segment de code.
           END                  ; Fin de la routine.
```

Après déclaration et affectation du segment de code (c'est lui qui renferme le code objet de la routine...), la routine d'addition est déclarée en procédure de type FAR. Notez à ce propos que le Fortran 77 de Microsoft n'admet que des noms de routines ne dépassant pas 6 caractères, ce qui nous a obligés à rogner de quelques caractères le nom de la routine ADDITION pour le transformer en ADD en début et en fin de déclaration de procédure.

Ceci étant, les lignes suivantes sauvegardent l'index source SI (on eut pu tout aussi bien employer l'index destinataire DI...) et le pointeur de base BP avant que ne soit attribué à ce dernier la valeur du pointeur de pile SP. Or, c'est cette valeur qui, mise dans BP, va nous servir à gérer les paramètres dans la pile.

Vous noterez que l'on ne transmet pas à la routine les valeurs numériques des paramètres, mais bien au contraire l'adresse où l'on est susceptible de les trouver et que ceux-ci s'empilent dans l'ordre inverse de leur écriture, le dernier paramètre transmis étant affecté du décalage minimum de 06. Ceci explique pourquoi le transfert des paramètres dans les registres AX et BX se passe en deux temps, l'index source recevant l'adresse dans un premier temps, adresse qui sert ensuite à transférer la valeur des paramètres d'entrée dans les registres AX et BX dans un second temps.

Dans ces conditions, le pas de décalage employé au sein de la routine pour lire la pile est de 4 octets, que les paramètres lui soient transmis sous la forme d'entiers sur deux octets ou de type simple ou double précision. Ne tombez pas dans le piège qui consiste à croire que le pas de décalage est dépendant du type de variable choisie : il n'en est rien !...

L'addition étant effectuée, le résultat placé dans AX, il ne reste plus qu'à établir dans leur valeur d'origine l'index source et le pointeur de base avant de rendre la main au programme d'appel. Notez au passage que le RET final est suivi du nombre 8, ce qui équivaut à 2 paramètres de 4 octets chacun et assure la gestion correcte de la pile.

Remarquez également que la routine doit être déclarée de type PUBLIC afin de pouvoir être reconnue par l'éditeur de liens. Dans le même ordre d'idées, il vous faut rajouter l'indication 'CODE' à la suite de la déclaration du segment de code, sans quoi l'éditeur de liens ne saurait où le situer.

Important : *Une routine écrite selon la procédure d'appel du FORTRAN 77 fonctionne parfaitement en TurboPascal 4.0, à condition de la considérer comme une procédure externe et de prévoir, dans la déclaration de cette procédure, un passage des paramètres sous forme d'adresses. Seule différence sans conséquence : le RET final.*

Programme d'essai en Fortran 77

La seconde méthode d'intégration d'une routine au Fortran consiste à considérer celle-ci comme une fonction externe. Or, cette méthode ne diffère de la précédente que par des points de détails, à savoir qu'il faut préalablement déclarer en Fortran le nom de la fonction comme s'il s'agissait d'une variable (entière dans le cas présent...) :

```
      PROGRAM ESSAI2
      INTEGER*2 B,C,ADD
      B=7
      C=ADD (5,B)
      WRITE (*,100) C
100   FORMAT (1X,'5+7=',I2)
      END
```

Vous noterez la déclaration de la fonction ADD en tant que variable entière (ligne n° 2) et l'appel direct de celle-ci sans passer par l'intermédiaire de l'instruction CALL (ligne n° 4). De surcroît, la procédure d'appel du Fortran autorise indifféremment le passage de valeurs numériques (le nombre 5 dans cet exemple...) et de variables (B et C en l'occurence...) à la routine d'addition sans déclaration préalable.

Procédure d'intégration au Fortran 77

Cela étant, il vous faut maintenant vérifier si cette méthode d'intégration de routines fonctionne correctement. Pour ce faire, vous allez taper la suite d'ordres que voici :

```
>MASM ADD;

>FOR1 ESSAI;

>PAS2

>LINK ESSAI ADD;
```

Pour mémoire, ces ordres correspondent respectivement à l'assemblage du code objet de la routine, aux 2 passes de compilation du Fortran et, pour finir, à l'édition de liens par laquelle on intègre le fichier .OBJ de la routine au code exécutable du programme.

A ce propos, notez qu'il est inutile de transformer le fichier .OBJ de la routine en une librairie d'extension .LIB, pourvu que l'on prenne la précaution de lancer l'édition de liens comme indiqué, c'est-à-dire en plaçant l'ensemble des commandes sur une même ligne.

Ceci fait, il vous reste à essayer le fichier ESSAI.EXE résultant. Normalement, vous devriez aboutir à quelque chose ressemblant à ceci :

```
>ESSAI
5+7=12
>
```

Récapitulatif de la procédure d'appel

Fortran 77 Fonction	
Instruction d'appel	aucune
Paramètres	variables et valeurs numériques
Transmis sous forme	d'adresses
Empilage dans l'ordre	inverse d'écriture
Procédure de type	FAR
Décalage initial	06H
Pas de décalage	4 octets
Fin de routine	RET n(*)
Directives à spécifier :	PUBLIC et 'CODE'

(*) n = Nombre de paramètres transmis × pas de décalage.

C MICROSOFT Version 4.0

Fonction

Nouveau venu parmi les langages de programmation, le C est le langage à la mode de ces dernières années, particulièrement prisé par les développeurs de logiciels graphiques. Finis les sous-programmes, procédures et autres fonctions des Basics, Pascals et Fortran. Le langage C regroupe tout cela sous la forme de fonctions aux possibilités élargies.

En effet, le C Microsoft est capable d'appeler des routines Assembleur par le biais de fonctions qui se comportent à la fois comme des fonctions ou des procédures, qu'il est inutile de déclarer, auxquelles on peut passer les paramètres sous forme d'adresses ou de valeurs et qui tolèrent indifféremment des variables ou des valeurs numériques. Que peut-on souhaiter de plus ?...

Routine en Assembleur

Afin d'illustrer ce qui va suivre, nous avons choisi une routine d'addition de 2 nombres entiers qui, bien que d'une utilité discutable (le C est déjà pourvu en la matière...), assure le contrôle du passage correct des paramètres sous forme de valeurs entre le programme et la routine, et vice-versa.

```
;          ******************************************
;          Routine d'addition adaptée au C Microsoft
;          ******************************************

CODE       SEGMENT BYTE 'CODE'   ; Déclaration du segment de code
           ASSUME  CS:CODE       ; Affectation du segment de code
           PUBLIC  _addition     ; Routine de type "public"
_addition  PROC    FAR           ; Déclaration de procédure FAR
                                 ;
           PUSH    BP            ; Sauvegarde pointeur de base
           MOV     BP,SP         ; Transfert pointeur de pile en BP
                                 ;
           MOV     AX,(BP+06H)   ; Chargement valeur de A en AX
           MOV     BX,(BP+08H)   ; Chargement valeur de B en BX
           ADD     AX,BX         ; Addition de A avec B (résultat en AX)
                                 ;
           POP     BP            ; Rétablissement pointeur de base
           RET                   ; ... et retour à l'envoyeur
                                 ;
_addition  ENDP                  ; Fin de la procédure d'addition
CODE       ENDS                  ; Fin du segment de code
           END                   ; Fin de la routine
```

Après déclaration et affectation du segment de code (c'est lui qui renferme le code objet de la routine...), la routine d'addition est déclarée en procédure de type FAR.

Ceci étant, les lignes suivantes sauvegardent le pointeur de base BP avant que ne soit attribué à ce dernier la valeur du pointeur de pile SP. Or, c'est cette valeur qui, mise dans BP, va nous servir à gérer les paramètres dans la pile.

Vous noterez que l'on transmet à la routine les paramètres sous forme de valeurs et que ceux-ci s'empilent dans l'ordre de leur écriture, le premier paramètre transmis étant affecté du décalage minimum de 06.

S'agissant d'entiers transmis sous forme de valeurs, le pas de décalage employé au sein de la routine pour lire la pile est de 2 octets, le premier paramètre étant à l'adresse 06H dans la pile et le second à l'adresse 08H.

L'addition étant effectuée et le résultat réacheminé automatiquement vers la variable C, il ne reste plus qu'à rétablir dans leur valeur d'origine le pointeur de base avant de rendre la main au programme d'appel. Notez au passage que le RET final est dépourvu de toute indication de dépilage (en nombre d'octets...), la procédure d'appel du C étant assez grande pour effectuer toute seule ce calcul.

Si maintenant on s'intéresse aux particularités d'écriture de la routine, on constate que le nom de celle-ci doit OBLIGATOIREMENT commencer par un trait de soulignement, sans quoi l'éditeur de liens ne saurait l'intégrer au fichier exécutable. Par contre, vous pouvez employer indifféremment des majuscules ou des minuscules dans le nom de la routine, même si le nom de la fonction qui lui correspond dans le programme C est écrit en minuscules.

Remarquez également que la routine doit être déclarée de type PUBLIC afin de pouvoir être reconnue par l'éditeur de liens. Dans le même ordre d'idées, il vous faut rajouter l'indication 'CODE' à la suite de la déclaration du segment de code, sans quoi l'éditeur de liens ne saurait où le situer.

Important : *La procédure au sein de la routine étant déclarée de type FAR, il vous faut OBLIGATOIREMENT utiliser l'option médium (/AM) lors de la compilation, sans quoi c'est la panique la plus totale chez l'éditeur de liens.*

Programme d'essai en C Microsoft

Cette première formulation d'écriture que nous examinons (c'est en fait la plus courante...) consiste à considérer la routine Assembleur comme une fonction usuelle, c'est-à-dire qu'après lui avoir transmis des paramètres, celle-ci en retour nous fourni une valeur numérique, comme dans l'exemple ci-dessous :

```
main()
{
  int b,c;
  b=7;
  c=addition(5,b);
  printf("5+7=%d\n",c);
}
```

Que dire de ce programme d'essai, si ce n'est qu'on peut difficilement faire plus court. Notez tout de même l'absence de déclaration de la fonction addition en tant que fonction externe, celle-ci l'étant par défaut.

Procédure d'intégration au C Microsoft

D'ailleurs, nous allons vérifier que tout cela fonctionne en tapant la suite d'ordres d'assemblage, de compilation et d'édition de liens que voici :

```
>MASM ADDITION;

>MSC ESSAI/AM;

>LINK ESSAI ADDITION;
```

Pour mémoire, ces ordres correspondent respectivement à l'assemblage du code objet de la routine d'addition, aux passes du compilateur Microsoft C et, pour finir, à l'édition de liens par laquelle on intègre le fichier .OBJ de la routine au code exécutable du programme..

Notez au passage l'emploi de l'option / AM lors de la compilation (il s'agit de l'option médium du langage C...), afin d'être en concordance avec la déclaration de type FAR qui est faite dans la routine d'addition.

Notez enfin, qu'il est inutile de transformer le fichier .OBJ de la routine en une librairie d'extension .LIB, pourvu que l'on prenne la précaution de lancer l'édition de liens comme indiqué, c'est-à-dire en plaçant l'ensemble des commandes sur une même ligne.

Ceci fait, il vous reste à essayer le fichier ESSAI.EXE résultant. Normalement, vous devriez aboutir à quelque chose ressemblant à ceci :

```
>ESSAI
5+7=12
```

Retour de plusieurs paramètres

Une fonction est bien pratique car elle réexpédie automatiquement vers le programme appelant une valeur. L'inconvénient, c'est que précisément, elle ne peut renvoyer qu'UNE SEULE valeur !... Si la routine est censée renvoyer plusieurs valeurs (comme dans le cas du pilotage d'une souris ou d'une tablette graphique...), la forme d'écriture que nous venons d'examiner ne convient pas.

Ce qu'il faut, c'est réexpédier les paramètres attendus vers le programme C en les passant sous forme d'adresses. **Cette règle est impérative.**

Pour ce faire, il suffit de réécrire le programme d'essai comme suit :

```
main()
{
  int b,c;
  b=7;
  addition(5,b,&c)
  printf("5+7=%d\n",c);
}
```

Voyez-vous une différence avec le programme précédent ?... Celle-ci est subtile et réside dans le passage du paramètre C à la routine par le biais de son pointeur d'adresse (caractère &), ce dernier paramètre étant alors renvoyé sous forme d'adresse et non plus sous forme de valeur. D'ailleurs, rien ne vous empêche de procéder de même pour l'ensemble des paramètres si le cœur vous en dit, le C Microsoft étant suffisamment souple de ce point de vue pour accepter les deux écritures.

Quant à la routine d'addition, elle se voit greffer 4 lignes supplémentaires, deux pour le retour du contenu du registre AX vers la variable C (on renvoie cette fois-ci une adresse et non plus une valeur...) et les deux autres pour la sauvegarde et le rétablissement de l'index source SI (qui sert précisément au chargement de cette adresse) :

```
CODE       SEGMENT BYTE 'CODE'    ; Déclaration du segment de code
           ASSUME  CS:CODE        ; Affectation du segment de code
           PUBLIC  _addition      ; Routine de type "public"
_addition  PROC    FAR            ; Déclaration de procédure FAR
                                  ;
           PUSH    BP             ; Sauvegarde pointeur de base
           MOV     BP,SP          ; Transfert pointeur de pile en BP
           PUSH    SI             ; Sauvegarde index source
                                  ;
           MOV     AX,(BP+06H)    ; Chargement valeur de A en AX
           MOV     BX,(BP+08H)    ; Chargement valeur de B en BX
           ADD     AX,BX          ; Addition de A avec B (résultat en AX)
           MOV     SI,(BP+0AH)    ; Chargement adresse de C en SI
           MOV     [SI],AX        ; Transfert contenu de AX à la variable C
                                  ;
           POP     SI             ; Rétablissement index source
           POP     BP             ; Rétablissement pointeur de base
           RET                    ; ... et retour à l'envoyeur
                                  ;
_addition  ENDP                   ; Fin de la procédure d'addition
CODE       ENDS                   ; Fin du segment de code
           END                    ; Fin de la routine
```

S'agissant des paramètres passés sous forme d'adresses. le nombre d'octets nécessaires au passage dépend de l'option choisie. Dans le cas de l'option médium, il est de 2 octets.

Récapitulatif de la procédure d'appel

C Microsoft version 4.0 Fonction		
Instruction d'appel	:	aucune
Paramètres		variables et valeurs numériques (1)
Transmis sous forme		de valeurs (1)
Empilage dans l'ordre		normal d'écriture
Procédure de type		FAR
Décalage initial		06H
Pas de décalage		(2 à 8 octets pour les nombres) (1)
Fin de routine		RET
Directives à spécifier	:	PUBLIC et 'CODE'

(1) Il est à noter que la procédure indiquée correspond au passage de paramètres sous forme de valeurs. Dans le cas où ceux-ci sont transmis sous forme d'adresses, le pas de décalage dépend de l'option choisie.

QUICK C version 1.0

Fonction

Dernier-né des langages C, le Quick C est la réplique de Microsoft au Turbo C de Borland. Possédant le même type d'environnement intégré de développement, il se distingue de ce dernier par une procédure d'intégration des librairies qui n'est pas sans rappeler celle du Quick Basic.

Comme le C Microsoft, il est capable d'appeler des routines Assembleur par le biais de fonctions qui se comportent à la fois comme des fonctions ou des procédures, qu'il est inutile de déclarer, auxquelles on peut passer les paramètres sous forme d'adresses ou de valeurs et qui tolèrent indifféremment des variables ou des valeurs numériques. Que peut-on souhaiter de plus ?...

Routine en Assembleur

Afin d'illustrer ce qui va suivre, nous avons choisi une routine d'addition de 2 nombres entiers qui, bien que d'une utilité discutable (le Quick C est déjà pourvu en la matière...), assure le contrôle du passage correct des paramètres sous forme de valeurs entre le programme et la routine, et vice-versa.

```
;          ************************************************
;          Routine d'addition appelable à partir du Quick C
;          ************************************************

CODE       SEGMENT BYTE 'CODE'   ; Déclaration du segment de code.
           ASSUME  CS:CODE       ; Affectation du segment de code.
           PUBLIC  _addition     ; Routine déclarée de type "public".
_addition  PROC    FAR           ; Déclaration de procédure FAR.
                                 ;
           PUSH    BP            ; Sauvegarde pointeur de base.
           MOV     BP,SP         ; Transfert pointeur de pile en BP.
                                 ;
           MOV     AX,(BP+Ø6H)   ; Transfert 1er paramètre en AX.
           MOV     BX,(BP+Ø8H)   ; Transfert 2ème paramètre en BX.
           ADD     AX,BX         ; Addition de A + B (résultat en AX).
                                 ;
           POP     BP            ; Rétablissement pointeur de base
           RET                   ; ... et retour à l'envoyeur.
                                 ;
_addition  ENDP                  ; Fin de procédure.
CODE       ENDS                  ; Fin de segment de code.
           END                   ; Fin de routine.
```

Après déclaration et affectation du segment de code (c'est lui qui renferme le code objet de la routine...), la routine d'addition est déclarée en procédure de type FAR.

Ceci étant, les 2 lignes suivantes sauvegardent le pointeur de base BP avant que ne soit attribué à ce dernier la valeur du pointeur de pile SP. Or, c'est cette valeur qui, mise dans BP, va nous servir à gérer les paramètres dans la pile.

Vous noterez que l'on transmet à la routine les paramètres sous forme de valeurs et que ceux-ci s'empilent dans l'ordre de leur écriture, le premier paramètre transmis étant affecté du décalage minimum de 06.

S'agissant d'entiers transmis sous forme de valeurs, le pas de décalage employé au sein de la routine pour lire la pile est de 2 octets, le premier paramètre étant à l'adresse 06H dans la pile et le second à l'adresse 08H.

L'addition étant effectuée et le résultat réacheminé automatiquement vers la variable C, il ne reste plus qu'à rétablir dans leur valeur d'origine le pointeur de base avant de rendre la main au programme d'appel. Notez au passage que le RET final est dépourvu de toute indication de dépilage (en nombre d'octets...), la procédure d'appel du Quick C étant assez grande pour effectuer toute seule ce calcul.

Si maintenant on s'intéresse aux particularités d'écriture de la routine, on constate que le nom de celle-ci doit OBLIGATOIREMENT commencer par un trait de soulignement, sans quoi l'éditeur de liens ne saurait l'intégrer au fichier exécutable.

Remarquez également que la routine doit être déclarée de type PUBLIC afin de pouvoir être reconnue par l'éditeur de liens. Dans le même ordre d'idées, il vous faut rajouter l'indication 'CODE' à la suite de la déclaration du segment de code, sans quoi l'éditeur de liens ne saurait où le situer.

Important : *Contrairement au C où l'on pouvait utiliser indifféremment des majuscules ou des minuscules dans les noms de routine, le Quick C n'accepte que des minuscules. En conséquence, il y aura lieu d'employer l'option /MX lors de l'assemblage.*

Programme d'essai en Quick C

Cette première formulation d'écriture que nous examinons (c'est en fait la plus courante...) consiste à considérer la routine Assembleur comme une fonction usuelle, c'est-à-dire qu'après lui avoir transmis des paramètres, celle-ci en retour nous fourni une valeur numérique, comme dans l'exemple ci-dessous.

```
main()
{
  int b,c;
  b=7;
  c=addition(5,b);
  printf("5+7=%d\n",c);
}
```

Que dire de ce programme d'essai, si ce n'est qu'on peut difficilement faire plus court. Notez tout de même l'absence de déclaration de la fonction addition en tant que fonction externe, celle-ci l'étant par défaut.

Procédure d'intégration au Quick C

La routine étant tapée sous un éditeur vous allez l'assembler avec le Macro-Assembleur MASM de Microsoft, en n'oubliant pas d'inclure l'option /MX de façon à conserver les minuscules telles quelles :

```
MASM ADDITION/MX;
```

Ceci fait, il vous faut maintenant l'incorporer à l'environnement intégré de développement du QuickBasic de façon à pouvoir l'appeler directement à partir de votre programme.

Pour ce faire, il vous faut créer une librairie de développement d'extension .QLB qui sera appelée à chaque compilation et édition de liens déclenchées par l'option RUN de l'environnement intégré.

Cette librairie, baptisée ESSAI.QLB pour la circonstance (bien que tout autre nom de votre choix convienne...), s'obtient en lançant comme suit l'utilitaire LINK :

```
>LINK QUICKLIB.OBJ+ADDITION.OBJ,ESSAI.QLB,,/Q;
```

Ceci fait, il vous suffit maintenant de procéder au chargement de la librairie ESSAI.QLB en même temps que le Quick C en incluant l'option /l (c'est un l minuscule...), suivie du nom complet de la librairie à la suite de l'ordre de lancement du Quick C :

```
>QC /l ESSAI.QLB
```

Important : *Notez l'emploi obligatoire d'un l minuscule (un L majuscule ne serait pas compris...) pour charger la librairie en même temps que le Quick C, associé à un espace (également obligatoire...) entre les lettres QC et l'option /l.*

Quant à l'obtention d'un fichier exécutable autonome, il nous a pas été donné d'y parvenir à partir de l'environnement intégré du Quick C. C'est pourquoi nous avons eu recours à la bonne vieille méthode de l'édition de liens séparée. Pour ce faire, vous sélectionnez sous l'environnement intégré les options « Compile » et ( )Obj dans le menu RUN de façon à obtenir un fichier .OBJ.

Ceci fait, il vous suffit de sortir de l'environnement intégré et de lancer comme suit l'éditeur de liens :

```
>LINK ESSAI ADDITION;
```

Vous obtenez ainsi un fichier .EXE autonome qui ne demande qu'à être essayé :

```
>ESSAI
5+7=12
>
```

Retour de plusieurs paramètres

Une fonction est bien pratique car elle réexpédie automatiquement vers le programme appelant une valeur. L'inconvénient, c'est que précisément, elle ne peut renvoyer qu'UNE SEULE valeur !... Si la routine est censée renvoyer plusieurs valeurs (comme dans le cas du pilotage d'une souris ou d'une tablette graphique...), la forme d'écriture que nous venons d'examiner ne convient pas.

Ce qu'il faut, c'est réexpédier les paramètres attendus vers le programme en les passant sous forme d'adresses. **Cette règle est impérative.**

Pour ce faire, il suffit de réécrire le programme d'essai comme suit :

```
main()
{
  int b,c;
  b=7;
  addition(5,b,&c)
  printf("5+7=%d\n",c);
}
```

Voyez-vous une différence avec le programme précédent ?... Celle-ci est subtile et réside dans le passage du paramètre C à la routine par le biais de son pointeur d'adresse (caractère &), ce dernier paramètre étant alors renvoyé sous forme d'adresse et non plus sous forme de valeur. D'ailleurs, rien ne vous empêche de procéder de même pour l'ensemble des paramètres si le cœur vous en dit, le Quick C étant suffisamment souple de ce point de vue pour accepter les deux écritures.

Quant à la routine d'addition, elle se voit greffer 4 lignes supplémentaires, deux pour le retour du contenu du registre AX vers la variable C (on renvoie cette fois-ci une adresse et non plus une valeur...) et les deux autres pour la sauvegarde et le rétablissement de l'index source SI (qui sert précisément au chargement de cette adresse) :

```
CODE      SEGMENT BYTE 'CODE'  ; Déclaration du segment de code
          ASSUME  CS:CODE      ; Affectation du segment de code
          PUBLIC  _addition    ; Routine de type "public"
_addition PROC    FAR          ; Déclaration de procédure FAR
                               ;
          PUSH    BP           ; Sauvegarde pointeur de base
          MOV     BP,SP        ; Transfert pointeur de pile en BP
          PUSH    SI           ; Sauvegarde index source
                               ;
          MOV     AX,(BP+06H)  ; Chargement valeur de A en AX
          MOV     BX,(BP+08H)  ; Chargement valeur de B en BX
          ADD     AX,BX        ; Addition de A avec B (résultat en AX)
          MOV     SI,(BP+0AH)  ; Chargement adresse de C en SI
          MOV     [SI],AX      ; Transfert contenu de AX à la variable C
                               ;
          POP     SI           ; Rétablissement index source
          POP     BP           ; Rétablissement pointeur de base
          RET                  ; ... et retour à l'envoyeur
                               ;
_addition ENDP                 ; Fin de la procédure d'addition
CODE      ENDS                 ; Fin du segment de code
          END                  ; Fin de la routine
```

S'agissant des paramètres passés sous forme d'adresses, le nombre d'octets nécessaires au passage dépend de l'option choisie. Dans le cas de l'option médium, il est de 2 octets.

Récapitulatif de la procédure d'appel

Quick C version 1.0 Fonction		
Instruction d'appel	:	aucune
Paramètres		variables et valeurs numériques (1)
Transmis sous forme		de valeurs (1)
Empilage dans l'ordre		normal d'écriture
Procédure de type		FAR
Décalage initial		06H
Pas de décalage		(2 à 8 octets pour les nombres) (1)
Fin de routine		RET
Directives à spécifier	:	PUBLIC et 'CODE'

(1) Il est à noter que la procédure indiquée correspond au passage de paramètres sous forme de valeurs. Dans le cas où ceux-ci sont transmis sous forme d'adresses, le pas de décalage dépend de l'option choisie.

TURBO C version 1.5

Assembleur en ligne

Concurrent direct du Quick C Microsoft, le Turbo C Borland (dans sa version 1.5) possède lui aussi un environnement intégré de développement ainsi qu'une remarquable librairie graphique. Il s'en distingue toutefois par une de ses procédures d'intégration qui consiste à écrire le code source de la routine directement en Assembleur au sein du programme.

La procédure d'appel du Turbo C, ce que la littérature anglo-saxonne nomme « The Turbo C Calling Procedure » est, à quelques détails près, celle du C et du Quick C Microsoft. Le seul point de divergence concerne l'emploi obligatoire de minuscules dans les noms de routines.

Pour le reste, le Turbo C est capable d'appeler des routines Assembleur par le biais de fonctions auxquelles on peut passer les paramètres sous forme d'adresses ou de valeurs et qui acceptent indifféremment des variables ou des valeurs numériques.

Routine en Assembleur

Afin d'illustrer ce qui va suivre, nous avons choisi une routine d'addition de 2 nombres entiers qui, bien que d'une utilité discutable (le Turbo C est déjà pourvu en la matière...), permet de contrôler le passage correct des paramètres entre le programme appelant et la routine, et vice-versa.

Son écriture est à faire directement dans le programme en Turbo C et, en l'occurrence, au début de celui-ci, comme dans l'exemple ci-après :

```
#pragma inline

int addition(int a,int b)
{
  asm  mov  ax,a
  asm  mov  bx,b
  asm  add  ax,bx
  return(_AX);
}
```

Vous noterez que ce début de programme commence par la directive de compilation #pragma inline qui, bien que non indispensable, permet au compilateur de traiter l'assembleur en ligne dès la première tentative de compilation.

Vient ensuite la fonction d'addition qui comporte 3 lignes d'assembleur en ligne débutant par l'instruction asm, suivies par l'indication du registre de retour (AX en l'occurrence...). A ce propos, il est bon de noter que le point virgule n'est pas indispensable à la fin des lignes d'assembleur en ligne et que l'indication du registre de retour se doit d'être écrite en majuscule précédée par un trait de soulignement.

Programme d'essai en Turbo C

La routine écrite (en Assembleur s'il vous plait...) en tête du programme d'essai, il ne reste plus qu'à terminer celui-ci pour qu'il puisse l'appeler comme une fonction :

```
main()
{
  int b,c;
  b=7;
  c=addition(5,b);
  printf("5+7=%d\n",c);
}
```

Ceci fait, il faut procéder à la compilation de cette fonction d'addition. Pour ce faire, il vous faut installer le Macro-assembleur Microsoft (version 3.0 ou plus récente) dans le même sous-répertoire que votre Turbo C car, lors de la compilation du programme, le compilateur C va passer la main au compilateur MASM pour la partie du programme écrit en Assembleur.

Ce chargement accompli, il ne reste plus qu'à sortir de l'environnement intégré du Turbo C et à lancer la compilation séparée du programme d'essai en tapant sous DOS :

```
>TCC ESSAI
```

Reste à essayer le fichier .EXE résultant de cette double compilation. Normalement, vous devriez obtenir quelque chose ressemblant à ceci :

```
>ESSAI
5+7=12
>
```

L'inconvénient d'une telle méthode, c'est qu'il faut procéder à une compilation séparée du programme, ce qui en alourdit considérablement la mise au point par les va-et-vient incessants qu'elle nécessite.

Récapitulatif de la procédure d'appel

Turbo C version 1.5 Assembleur en ligne		
Instruction d'appel	:	aucune
Paramètres	:	variables ou valeurs numériques
Transmis sous forme	:	affectation directe
Empilage dans l'ordre	:	affectation directe
Procédure de type	:	Inutile
Décalage initial	:	Inutile
Pas de décalage	:	Inutile
Fin de routine	:	return (—AX) (*)
Directives à spécifier	:	aucune

(*) ou tout autre registre de votre choix.

TURBO C version 1.5

Fonction

Le Turbo C, au même titre que le C et le Quick C Microsoft, propose de reconnaître une routine Assembleur comme s'il s'agissait d'une fonction externe aux possibilités élargies regroupant toutes les fonctionnalités propres aux fonctions, sous-programmes et procédures des autres langages.

La procédure d'appel du Turbo C, ce que la littérature anglo-saxonne nomme « The Turbo C Calling Procedure » est, à quelques détails près, celle du C et du Quick C Microsoft. Le seul point de divergence concerne l'emploi obligatoire de minuscules dans les noms de routines.

Pour le reste, le Turbo C est capable d'appeler des routines Assembleur par le biais de fonctions auxquelles on peut passer les paramètres sous forme d'adresses ou de valeurs et qui acceptent indifféremment des variables ou des valeurs numériques.

Routine en Assembleur

Afin d'illustrer ce qui va suivre, nous avons choisi une routine d'addition de 2 nombres entiers qui, bien que d'une utilité discutable (le Turbo C est déjà pourvu en la matière...), permet de contrôler le passage correct des paramètres entre le programme appelant et la routine, et vice-versa.

```
;           ****************************************
;           Routine d'addition adaptée au Turbo C
;           ****************************************

CODE        SEGMENT BYTE 'CODE'  ; Déclaration du segment de code
            ASSUME  CS:CODE      ; Affectation du segment de code
            PUBLIC  _addition    ; Routine de type "public"
_addition   PROC    FAR          ; Déclaration de procédure FAR
                                 ;
            PUSH    BP           ; Sauvegarde pointeur de base
            MOV     BP,SP        ; Transfert pointeur de pile en BP
                                 ;
            MOV     AX,(BP+06H)  ; Chargement valeur de A en AX
            MOV     BX,(BP+08H)  ; Chargement valeur de B en BX
            ADD     AX,BX        ; Addition de A avec B (résultat en AX)
                                 ;
            POP     BP           ; Rétablissement pointeur de base
            RET                  ; ... et retour à l'envoyeur
                                 ;
_addition   ENDP                 ; Fin de la procédure d'addition
CODE        ENDS                 ; Fin du segment de code
            END                  ; Fin de la routine
```

Après déclaration et affectation du segment de code (c'est lui qui renferme le code objet de la routine...), la routine d'addition est déclarée en procédure de type FAR.

Ceci étant, les lignes suivantes sauvegardent le pointeur de base BP avant que ne soit attribué à ce dernier la valeur du pointeur de pile SP. Or, c'est cette valeur qui, mise dans BP, va nous servir à gérer les paramètres dans la pile.

Vous noterez que l'on transmet à la routine les paramètres sous forme de valeurs et que ceux-ci s'empilent dans l'ordre de leur écriture, le premier paramètre transmis étant affecté du décalage minimum de 06.

S'agissant d'entiers transmis sous forme de valeurs, le pas de décalage employé au sein de la routine pour lire la pile est de 2 octets, le premier paramètre étant à l'adresse 06H dans la pile et le second à l'adresse 08H.

L'addition étant effectuée et le résultat réacheminé automatiquement vers la variable C, il ne reste plus qu'à rétablir dans leur valeur d'origine le pointeur de base avant de rendre la main au programme d'appel. Notez au passage que le RET final est dépourvu de toute indication de dépilage (en nombre d'octets...), la procédure d'appel du C étant assez grande pour effectuer toute seule ce calcul.

S'agissant du nom de la routine, il faut OBLIGATOIREMENT que celui-ci débute par un trait de soulignement et qu'il soit écrit en minuscules, sans quoi l'éditeur de liens ne saurait l'intégrer au fichier exécutable.

Cette contrainte d'écriture implique donc, si l'on cherche pour des raisons de portabilité à développer un programme universel qui fonctionne indifférem-ment en Turbo C, Quick C et C, à systématiser l'écriture des noms de routine en minuscules et à assembler les dites routines en employant l'option /MX.

Remarquez également que la routine doit être déclarée de type PUBLIC afin de pouvoir être reconnue par l'éditeur de liens. Dans le même ordre d'idées, il vous faut rajouter l'indication 'CODE' à la suite de la déclaration du segment de code, sans quoi l'éditeur de liens ne saurait où le situer.

Programme d'essai en Turbo C

Cette première formulation d'écriture que nous examinons (c'est en fait la plus courante...) consiste à considérer la routine Assembleur comme une fonction, c'est-à-dire qu'après lui avoir transmis des paramètres, celle-ci en retour nous fourni une valeur numérique, comme dans l'exemple ci-dessous :

```
main()
{
  int b,c;
  b=7;
  c=addition(5,b);
  printf("5+7=%d\n",c);
}
```

Procédure d'intégration au Turbo C

Routine et programme étant écrits, il vous faut intégrer la première au second et, tant qu'à faire, dans l'environnement intégré de développement.

Pour ce faire, vous allez assembler la routine avec le macro-Assembleur MASM de Microsoft, en n'oubliant pas d'inclure l'option /MX pour conserver les minuscules telles quelles :

```
MASM ADDITION/MX;
```

Ceci fait, il vous faut créer le fichier projet ESSAI.PRJ qui va comporter les divers fichiers à lier ensemble lors de l'édition de liens et qui devrait ressembler à ceci :

```
ESSAI.C
ADDITION.OBJ
```

Ce fichier projet constitué et sauvegardé (ce qui peut être fait sous l'éditeur du Turbo C), il vous faut indiquer le nom de ce fichier dans l'option PROJECT, recharger le programme d'essai dans l'éditeur et l'exécuter en sélectionnant l'option RUN, un point c'est tout.

Pour le reste, à savoir l'obtention d'un module exécutable autonome, il suffit de sélectionner l'option « Make EXE file » dans le menu « Compile » pour que, après compilation et édition de liens, vous obteniez sur disquette ou disque dur un fichier .EXE autonome.

Retour de plusieurs paramètres

Une fonction est bien pratique car elle réexpédie automatiquement vers le programme appelant une valeur. L'inconvénient, c'est que précisément, elle ne peut renvoyer qu'UNE SEULE valeur !... Si la routine est censée renvoyer plusieurs valeurs (comme dans le cas du pilotage d'une souris ou d'une tablette graphique...), la forme d'écriture que nous venons d'examiner ne convient pas.

Ce qu'il faut, c'est réexpédier les paramètres attendus vers le programme en les passant sous forme d'adresses. **Cette règle est impérative.**

Pour ce faire, il suffit de réécrire le programme d'essai comme suit :

```
main()
{
  int b,c;
  b=7;
  addition(5,b,&c)
  printf("5+7=%d\n",c);
}
```

Par rapport au précédent, ce programme présente 2 différences. La première concerne l'absence de déclaration de fonction externe (en fait, ce n'en est plus vraiment une puisqu'elle ne renvoie aucune valeur...). La seconde est plus subtile et réside dans le passage du paramètre C à la routine par le biais de son pointeur d'adresse (caractère &), ce dernier paramètre étant alors renvoyé sous forme d'adresse et non plus sous forme de valeur. D'ailleurs, rien ne vous empêche de procéder de même pour l'ensemble des paramètres si le cœur vous en dit, le Turbo C étant suffisamment souple de ce point de vue pour accepter les deux écritures.

Quant à la routine d'addition, elle se voit greffer 4 lignes supplémentaires, deux pour le retour du contenu du registre AX vers la variable C (on renvoie cette fois-ci une adresse et non plus une valeur...) et les deux autres pour la sauvegarde et le rétablissement de l'index source SI (qui sert précisément au chargement de cette adresse) :

Important : ***L'option de compilation « small, médium, large, etc... » est sans effet sur le passage des paramètres tant que ceux-ci sont passés sous forme de valeurs. Il n'en est plus de même lorsqu'ils le sont sous forme d'adresses. A titre d'exemple, l'option médium s'accompagne d'un passage d'adresses sur 2 octets.***

```
CODE      SEGMENT BYTE 'CODE'   ; Déclaration du segment de code
          ASSUME  CS:CODE       ; Affectation du segment de code
          PUBLIC  _addition     ; Routine de type "public"
_addition PROC    FAR           ; Déclaration de procédure FAR
                                ;
          PUSH    BP            ; Sauvegarde pointeur de base
          MOV     BP,SP         ; Transfert pointeur de pile en BP
          PUSH    SI            ; Sauvegarde index source
                                ;
          MOV     AX,(BP+Ø6H)   ; Chargement valeur de A en AX
          MOV     BX,(BP+Ø8H)   ; Chargement valeur de B en BX
          ADD     AX,BX         ; Addition de A avec B (résultat en AX)
          MOV     SI,(BP+ØAH)   ; Chargement adresse de C en SI
          MOV     [SI],AX       ; Transfert contenu de AX à la variable C
                                ;
          POP     SI            ; Rétablissement index source
          POP     BP            ; Rétablissement pointeur de base
          RET                   ; ... et retour à l'envoyeur
                                ;
_addition ENDP                  ; Fin de la procédure d'addition
CODE      ENDS                  ; Fin du segment de code
          END                   ; Fin de la routine
```

S'agissant des paramètres passés sous forme d'adresses, le nombre d'octets nécessaires au passage dépend de l'option choisie. Dans le cas de l'option médium, il est de 2 octets.

Récapitulatif de la procédure d'appel

Turbo C version 1.5 Fonction	
Instruction d'appel	aucune
Paramètres	variables et valeurs numériques (1)
Transmis sous forme	de valeurs (1)
Empilage dans l'ordre	normal d'écriture
Procédure de type	FAR
Décalage initial	06H
Pas de décalage	(2 à 8 octets pour les nombres) (1)
Fin de routine	RET
Directives à spécifier	PUBLIC et 'CODE'

(1) Dans le cas où les paramètres sont transmis sous forme d'adresses, le pas de décalage dépend de l'option choisie.

30 routines Assembleur

Regroupées par thème, selon le périphérique auquel elles donnent accès, les routines suivantes vont vous permettre de :

Clavier/écran

Horloge

Haut-parleur

Carte graphique EGA

Disques et fichiers

Système

Souris

Port série

Tablette graphique

Imprimante

(*) XOR : mode qui permet de déplacer des entités (points, traits, zones coloriées, icônes, etc.) sur l'écran sans altérer le fond d'image.

Important

Que ceux qui auraient court-circuité la première partie de cet ouvrage veuillent bien lire les quelques lignes qui suivent (si tant est qu'ils ne court-circuitent pas également cette page...).

Les procédures d'appel des routines Assembleur sont aussi diversifiées qu'il existe de langages évolués. Il s'en suit qu'une routine écrite pour fonctionner avec un langage donné ne fonctionnera pas avec un autre.

Pour y parvenir néanmoins, il faudra modifier un certain nombre de lignes concernant :

- *les déclarations*
- *le passage des paramètres vers la routine*
- *le retour des paramètres*
- *la fin de routine*

C'est le rôle de la première partie de cet ouvrage que de vous expliquer, par le détail, les modifications qu'il convient d'apporter à une routine Assembleur pour qu'elle puisse fonctionner avec un langage plutôt qu'un autre.

Cela étant, il n'est pas possible dans cette seconde partie de l'ouvrage de vous présenter 12 versions différentes (une par langage...) de chaque routine. Plusieurs ouvrages y suffiraient à peine...

C'est pourquoi nous avons choisi de vous les présenter sous une écriture unique, celle qui convient au :

Turbo C

A vous d'effectuer les modifications qui s'imposent pour les adapter à votre langage de développement, aidé en cela par la première partie.

1

VIDER LA MÉMOIRE-TAMPON DU CLAVIER

Il est intéressant, voire indispensable dans certains cas, de pouvoir vider la mémoire-tampon du clavier avant d'appeler une instruction de saisie de caractères (du genre INPUT par exemple...). Or, il n'existe dans aucun des langages d'instruction capable de le faire...

Routine en Assembleur

```
;         **************************************************
;         Routine keydump.asm appelable à partir du Turbo C
;         **************************************************

CODE      SEGMENT BYTE 'CODE'
          ASSUME  CS:CODE
          PUBLIC  _keydump
_keydump  PROC    FAR

          PUSH    BP              ; Sauvegarde pointeur de base.
          MOV     BP,SP           ; Transfert pointeur de pile en BP.
                                  ;
          MOV     AX,0C00H        ; Fonction 0C (AH=0C) de
          INT     21H             ; l'interruption 21 du Bios.
                                  ;
          POP     BP              ; Rétablissement pointeur de base
          RET                     ; ... et retour à l'envoyeur.

_keydump  ENDP
CODE      ENDS
          END
```

Baptisée keydump pour la circonstance, cette routine fait appel à la fonction 0C de l'interruption 21 du Bios qui, précisément, a pour rôle de vider la mémoire-tampon du clavier.

Notez que le numéro de la fonction (0C en l'occurrence...) se doit de figurer dans le registre AH lors de l'appel de l'interruption 21 du Bios.

Programme d'essai en Turbo C

Réduit à sa plus simple expression, le programme d'essai de la fonction keydump se résume à ceci :

```
main()
{
  keydump();
}
```

Notez, toutefois, la présence obligatoire des parenthèses à la suite de la fonction keydump même si, comme c'est le cas, cette dernière ne transmet aucun paramètre à la routine. Cela fait partie de la règle du jeu avec le Turbo C (et les autres langages C également...).

2

OBTENIR LE CODE ASCII D'UNE TOUCHE

Voici un petit utilitaire très pratique qui vous permet d'obtenir le (ou les) codes ASCII des touches du clavier. Il suffit, après l'avoir lancé, d'appuyer sur une touche pour voir s'afficher le code ASCII de cette touche et, lorsqu'il s'agit des touches de fonctions et du pavé numérique, les deux codes ASCII qui leur correspondent.

Routine en Assembleur

```
;         *************************************************
;         Routine testkey.asm appelable à partir du Turbo C
;         *************************************************

CODE      SEGMENT BYTE 'CODE'
          ASSUME  CS:CODE
          PUBLIC  _testkey
_testkey  PROC    FAR

          PUSH    BP            ; Sauvegarde pointeur de base.
          MOV     BP,SP         ; Transfert pointeur de pile en BP.
                                ;
          MOV     AH,Ø8H        ; Saisie du caractère sans écho
          INT     21H           ; (fonction Ø8 interruption 21).
          XOR     AH,AH         ; RAZ de AH (code ASCII en AL).
                                ;
          POP     BP            ; Rétablissement pointeur de base
          RET                   ; ... et retour à l'envoyeur.

_testkey  ENDP
CODE      ENDS
          END
```

Baptisée testkey, cette routine fait appel à la fonction 08 de l'interruption 21 du Bios pour la saisie d'un caractère au clavier sans écho sur l'écran.

Notez que la valeur du code ASCII figure dans la partie basse (AL) du registre AX au retour de l'interruption 21 et que, dans ces conditions, il convient de nettoyer la partie haute (AH) de ce registre avant d'en réexpédier automatiquement le contenu vers le programme appelant.

Programme d'essai en Turbo C

```
main()
{
  int ascii;
  do
  {
    printf("\nAppuyez sur une touche...\n");
    ascii=testkey();
    if (ascii==0)
    {
      ascii=testkey();
      printf("1er code ASCII=0\n");
      printf("2nd code ASCII=%d\n",ascii);
    }
    else
    printf("code ASCII=%d\n",ascii);
  }
  while (ascii!=0);
```

Que dire de ce programme d'essai si ce n'est que lorsque le test porte sur une touche à code ASCII étendu, la routine testkey renvoie le code 0. Il faut donc dans ce cas appeler une seconde fois la fonction testkey pour obtenir le second octet du code étendu.

Quant à sortir du programme, il vous faut appuyer sur Ctrl-C ou Ctrl-Break. Pour votre information, cette sortie de programme fait partie intégrante de la fonction 08 de l'interruption 21. C'est pourquoi vous ne trouverez rien à ce sujet, ni dans le programme, ni dans la routine.

Conseil pratique

Si la routine vous plaît, constituez-vous donc un utilitaire de type .EXE que vous pourrez appeler sous DOS quand bon vous semble, à la manière des utilitaires du MS DOS.

3

ACTIVER LA TOUCHE CAPS LOCK

Combien de fois n'avez-vous souhaité à la mise sous tension de l'ordinateur de pouvoir activer automatiquement la touche Caps Lock de passage en majuscules. Désormais, c'est chose faisable, puisque la routine qui vous est proposée peut être appelée directement à partir du fichier AUTOEXEC.BAT.

Routine en Assembleur

```
;       ******************************************
;       Routine capson.asm écrite pour fichier .COM
;       ******************************************

        ORG     100H

CODE    SEGMENT

        MOV     AX,0000H     ; Chargement segment
        MOV     DS,AX        ; de code avec 0000.
        MOV     AL,01000000B ; Passage en capslock
        OR      DS:0417H,AL  ; (forçage bit n°6 à 1).
        RET                  ; Retour au DOS.

CODE    ENDS
        END
```

L'octet des touches spéciales du clavier (dont Caps Lock fait partie...) figure à l'adresse 0417 du segment 0000. Or, le bit n° 6 reflète l'état de cette touche (0 = non active, 1 = active).

Dans ces conditions, il suffit de forcer à 1 ce bit n° 6 si l'on veut passer en Caps Lock, ce que réalise fort bien la routine avec un OR et le masque 01000000 (40 en hexadécimal).

Point intéressant : le passage en Caps Lock s'accompagne de l'allumage du voyant situé sur la touche (sur AT uniquement).

Création d'un fichier .COM

Destiné à devenir autonome sous la forme d'un fichier .COM, cette routine débute par l'inévitable ORG 100H.

Remarquez l'absence d'affectation du segment de code, de déclaration de type public et de procédure FAR, avec ce que cela implique de sauvegarde du pointeur de base et de transfert du pointeur de pile en BP.

Ici, tout cela est inutile !... Il suffit de déclarer le segment de code, un point c'est tout !

Quant à la procédure qui permet d'obtenir un fichier .COM à partir du code source de la routine (.ASM), il suffit de lancer comme suit les 3 utilitaires que voici :

```
>MASM CAPSON;

>LINK CAPSON;

>EXE2BIN CAPSON CAPSON.COM
```

Le résultat est un fichier .COM de 12 octets de long qui ne demande qu'à être incorporé à un fichier AUTOEXEC.BAT.

4

DÉSACTIVER LA TOUCHE CAPS LOCK

Dans le même ordre d'idées que la routine précédente, il peut être intéressant de pouvoir désactiver la touche Caps Lock, ne serait-ce qu'à la sortie de certaines applications qui laissent cette touche active. C'est ce que vous propose la routine capsoff ci-après qui, en prime, éteint le voyant de la touche.

Routine en Assembleur

```
;       *********************************************
;       Routine capsoff.asm écrite pour fichier .COM
;       *********************************************

        ORG     100H

CODE    SEGMENT

        MOV     AX,0000H    ; Chargement segment
        MOV     DS,AX       ; de code avec 0000.
        MOV     AL,10111111B ; Désactivation de CapsLock
        AND     DS:0417H,AL ; (forçage bit n°6 à 0).
        RET                 ; Retour au DOS.

CODE    ENDS
        END
```

L'octet des touches spéciales du clavier (dont Caps Lock fait partie...) figure à l'adresse 0417 du segment 0000. Or, le bit n° 6 reflète l'état de cette touche (0 = non active, 1 = active).

Dans ces conditions, il suffit de forcer à 0 ce bit n° 6 si l'on veut désactiver Caps Lock, ce que réalise fort bien la routine avec un AND et le masque 10111111 (BF en hexadécimal).

Point intéressant : la désactivation de Caps Lock s'accompagne de l'extinction du voyant situé sur la touche (sur AT uniquement).

Création d'un fichier .COM

Destinée à devenir autonome sous la forme d'un fichier .COM, cette routine débute par l'inévitable ORG 100H.

Remarquez l'absence d'affectation du segment de code, de déclaration de type public et de procédure FAR, avec ce que cela implique de sauvegarde de pointeur de base et de transfert du pointeur de pile en BP.

Ici, tout cela est inutile !... Il suffit de déclarer le segment de code, un point c'est tout !

Quant à la procédure qui permet d'obtenir un fichier .COM à partir du code source de la routine (.ASM), il suffit de lancer comme suit les 3 utilitaires que voici :

```
>MASM CAPSOFF;

>LINK CAPSOFF;

>EXE2BIN CAPSOFF CAPSOFF.COM
```

Le résultat est un fichier .COM de 12 octets de long qui ne demande qu'à être incorporé à un fichier AUTOEXEC.BAT.

5

ACTIVER LA TOUCHE NUM LOCK

N'avez-vous jamais souhaité activer automatiquement la touche Num Lock à la mise sous tension de l'ordinateur ?... Si, n'est-ce pas ?... Or, il n'existe aucun ordre qui sache le faire. C'est pourquoi vous apprécierez la petite routine que voici qui, intégrée à un fichier .BAT, vous évitera toute manœuvre de cette touche.

Routine en Assembleur

```
;          *******************************************
;          Routine numon.asm écrite pour fichier .COM
;          *******************************************

           ORG      100H

CODE       SEGMENT

           MOV      AX,0000H        ; Chargement segment
           MOV      DS,AX           ; de code avec 0000.
           MOV      AL,00100000B    ; Activation de NumLock
           OR       DS:0417H,AL     ; (forçage bit n°5 à 1).
           RET                      ; Retour au DOS.

CODE       ENDS
           END
```

L'octet des touches spéciales du clavier (dont Num Lock fait partie...) figure à l'adresse 0417 du segment 0000. Or, le bit n° 5 reflète l'état de cette touche (0 = non active, 1 = active).
Dans ces conditions, il suffit de forcer à 1 ce bit n° 5 si l'on veut passer en Num Lock, ce que réalise fort bien la routine avec un OR et le masque 0010000 (20 en hexadécimal).

Point intéressant : le passage en Num Lock s'accompagne de l'allumage du voyant situé sur la touche (sur AT uniquement).

Création d'un fichier .COM

Destiné à devenir autonome sous la forme d'un fichier .COM, cette routine débute par l'inévitable ORG 100H.

Remarquez l'absence d'affectation du segment de code, de déclaration de type public et de procédure FAR, avec ce que cela implique de sauvegarde du pointeur de base et de transfert du pointeur de pile en BP.

Ici, tout cela est inutile !... Il suffit de déclarer le segment de code, un point c'est tout !

Quant à la procédure qui permet d'obtenir un fichier .COM à partir du code source de la routine (.ASM), il suffit de lancer comme suit les 3 utilitaires que voici :

```
>MASM NUMON;

>LINK NUMON;

>EXE2BIN NUMON NUMON.COM
```

Le résultat est un fichier .COM de 12 octets de long qui ne demande qu'à être incorporé à un fichier .BAT.

6

DÉSACTIVER LA TOUCHE NUM LOCK

Vous qui travaillez sur un clavier 102 touches, ne vous est-il jamais arrivé de souhaiter pouvoir désactiver automatiquement la touche Num Lock à la mise sous tension de l'ordinateur ?... Nous, si !... C'est pourquoi nous vous proposons la petite routine que voici (elle ne fait que 12 octets de long...) de façon à ce que vous puissiez l'incorporer dans votre fichier AUTOEXEC.BAT.

Routine en Assembleur

```
;          *******************************************
;          Routine numoff.asm écrite pour fichier .COM
;          *******************************************

           ORG     100H

CODE       SEGMENT

           MOV     AX,0000H       ; Chargement segment
           MOV     DS,AX          ; de code avec 0000.
           MOV     AL,11011111B   ; Désactivation de NumLock
           AND     DS:0417H,AL    ; (forçage bit n°5 à 0).
           RET                    ; Retour au DOS.

CODE       ENDS
           END
```

L'octet des touches spéciales du clavier (dont Num Lock fait partie...) figure à l'adresse 0417 du segment 0000. Or, le bit n° 5 reflète l'état de cette touche (0 = non active, 1 = active).

Dans ces conditions, il suffit de forcer à 0 ce bit n° 5 si l'on veut désactiver Num Lock, ce que réalise fort bien la routine avec un AND et le masque 11011111 (DF en hexadécimal).

Point intéressant : la désactivation de Num Lock s'accompagne de l'extinction du voyant situé sur la touche (sur AT uniquement).

Création d'un fichier .COM

Destiné à devenir autonome sous la forme d'un fichier .COM, cette routine débute par l'inévitable ORG 100H.

Remarquez l'absence d'affectation du segment de code, de déclaration de type public et de procédure FAR, avec ce que cela implique de sauvegarde du pointeur de base et de transfert du pointeur de pile en BP.

Ici, tout cela est inutile !... Il suffit de déclarer le segment de code, un point c'est tout !

Quant à la procédure qui permet d'obtenir un fichier .COM à partir du code source de la routine (.ASM), il suffit de lancer comme suit les 3 utilitaires que voici :

```
>MASM NUMOFF;

>LINK NUMOFF;

>EXE2BIN NUMOFF NUMOFF.COM
```

Le résultat est un fichier .COM de 12 octets de long qui ne demande qu'à être incorporé à un fichier AUTOEXEC.BAT.

7

LIRE L'ÉTAT DES TOUCHES SPÉCIALES

Afficher l'état des touches Ins, Caps Lock, Num Lock et Scroll Lock au bas d'un masque d'écran de saisie de données ne peut que faciliter celle-ci. Or, aucun langage évolué ne propose d'instruction sachant lire l'état de ces 4 touches spéciales. C'est pourquoi vous apprécierez la routine keystate que voici.

Routine en Assembleur

```
;          ***************************************************
;          Routine keystate.asm appelable à partir du Turbo C
;          ***************************************************

CODE       SEGMENT BYTE 'CODE'
           ASSUME  CS:CODE
           PUBLIC  _keystate
_keystate  PROC    FAR

           PUSH    BP            ; Sauvegarde pointeur de base.
           MOV     BP,SP         ; Transfert pointeur de pile en BP.
           PUSH    SI            ; Sauvegarde index source.
           PUSH    ES            ; Sauvegarde extra segment.
                                 ;
           MOV     AX,0000H      ; Transfert segment 0040
           MOV     ES,AX         ; dans l'extra segment.
                                 ;
           MOV     AL,ES:0417H   ; Lecture octet d'état des touches.
           AND     AL,10000000B  ; Test touche Insert (bit n°7).
           MOV     SI,(BP+06H)   ; Adresse de la variable caps.
           MOV     [SI],AX       ; Retour état touche Capslock.
```

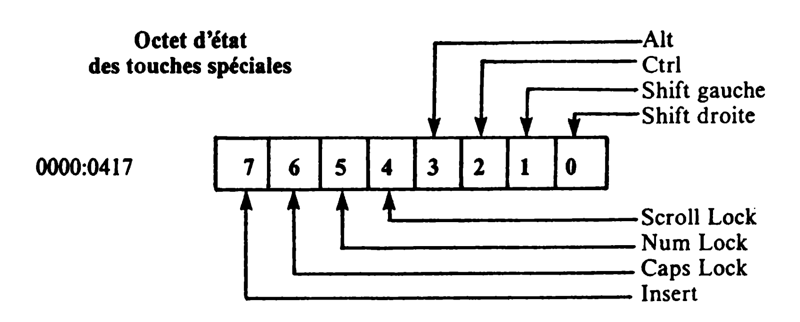
Routine en Assembleur (suite)

```
                                    ;
        MOV     AL,ES:0417H         ; Lecture octet d'état des touches.
        AND     AL,01000000B        ; Test touche Capslock (bit n°6).
        MOV     SI,(BP+08H)         ; Adresse de la variable caps.
        MOV     [SI],AX             ; Retour état touche Capslock.
                                    ;
        MOV     AL,ES:0417H         ; Lecture octet d'état des touches.
        AND     AL,00100000B        ; Test touche Numlock (bit n°5).
        MOV     SI,(BP+0AH)         ; Adresse de la variable num.
        MOV     [SI],AX             ; Retour état touche Numlock.
                                    ;
        MOV     AL,ES:0417H         ; Lecture octet d'état des touches.
        AND     AL,00010000B        ; Test touche Scroll (bit n°4).
        MOV     SI,(BP+0CH)         ; Adresse de la variable ins.
        MOV     [SI],AX             ; Retour état touche Insert.
                                    ;
        POP     ES                  ; Rétablissement extra segment.
        POP     SI                  ; Rétablissement index source.
        POP     BP                  ; Rétablissement pointeur de base
        RET                         ; ... et retour à l'envoyeur.

_keystate ENDP
CODE      ENDS
          END
```

L'octet d'état des touches spéciales du clavier figurant à l'adresse 0417 du segment 0000, il convient de charger l'extra-segment (on eut pu tout aussi bien employer le segment de données DS...) avec la valeur 0.

Ce faisant, il suffit alors de charger l'octet d'état des touches spéciales dans le registre AL et d'effectuer un ET logique avec le bit correspondant à la touche interrogée :



On sait alors, selon le résultat de cette opération, si la touche interrogée est active ou non (1 = active 0 = inactive).

Programme d'essai en Turbo C

```
main()
{
  int ins,caps,num,scroll;
  do
  {
    keystate(&ins,&caps,&num,&scroll);

    locate(25,58);
    if(ins==0)
      printf("       ");
    else
      printf("Insert ");

    if(caps==0)
      printf("     ");
    else
      printf("CAPS ");

    if(num==0)
      printf("    ");
    else
      printf("NUM ");

    if(scroll==0)
      printf("      ");
    else
      printf("SCROLL");
  }
  while(caps!=2);
}
```

Après déclaration des variables, la fonction keystate récupère l'état des 4 touches spéciales. Notez l'emploi du caractère & qui signale une transmission des paramètres sous forme d'adresse (cf première partie de l'ouvrage).

Cela étant, le programme affiche ou efface le nom de chaque touche selon qu'elle est active ou non. Notez à ce propos l'emploi de la routine locate pour positionner les indications des touches dans le coin inférieur droit de l'écran.

8

POSITIONNER LE CURSEUR

Certains langages étant démunis d'instruction de positionnement du curseur (notamment le FORTRAN et les Cs...), voici une petite routine qui comble cette lacune.

Routine en Assembleur

```
;        ************************************************
;        Routine locate.asm appelable à partir du Turbo C
;        ************************************************

CODE     SEGMENT BYTE 'CODE'
         ASSUME  CS:CODE
         PUBLIC   _locate
_locate  PROC    FAR

         PUSH    BP              ; Sauvegarde pointeur de base.
         MOV     BP,SP           ; Transfert pointeur de pile en BP.
                                 ;
         MOV     AX,(BP+Ø6H)     ; Chargement n° de ligne (1 à 25).
         DEC     AL              ; Décrémentation n° de ligne
         MOV     DH,AL           ; et transfert en DH.
                                 ;
         MOV     AX,(BP+Ø8H)     ; Chargement n° de colonne (1 à 8Ø).
         DEC     AL              ; Décrémentation n° de colonne
         MOV     DL,AL           ; et transfert en DL.
```

Routine en Assembleur (suite)

```
                                    ;
          XOR      BH,BH            ; Page active n° Ø en BH.
          MOV      AH,Ø2H           ; Positionnement du curseur
          INT      1ØH              ; (fonction Ø2 interruption 1Ø).
                                    ;
          POP      BP               ; Rétablissement pointeur de base
          RET                       ; ... et retour à l'envoyeur.

_locate   ENDP
CODE      ENDS
          END
```

Faisant appel à la fonction 02 de l'interruption 10 du Bios pour positionner le curseur, il convient de précharger les registres AH, BH, DH et DL avec les valeurs suivantes :

AH = 02 (c'est le n° de la fonction)

BH = n° de la page active

DH = n° de ligne (0 à 24)

DL = n° de colonne (0 à 79)

Cette routine locate étant calquée sur l'instruction LOCATE des Basics, nous avons volontairement conservée la numérotation des lignes et des colonnes de cette dernière, à savoir :

locate *(ligne, colonne)* avec { ligne = 1 à 25 ; colonne = 1 à 80 }

C'est pourquoi l'on décrémente ligne et colonne après leur chargement en AL et avant leur transfert en DH/DL.

Programme d'essai en Turbo C

```
main()
{
  locate(12,28);
  printf("Essai de positionnement...");
}
```

Court, mais parlant, ce programme d'essai se charge, après appel de la fonction locate, de positionner un texte en le faisant débuter sur la ligne 12, colonne 28.

9

AFFICHER 43 LIGNES DE TEXTE

Pour peu que vous disposiez d'une carte EGA, il vous est désormais possible d'afficher 43 lignes de texte (et non plus 25) sur l'écran. Il suffit pour cela de lancer la routine ega43 que nous vous proposons ci-après.

Routine en Assembleur

```
;       *******************************************
;       Routine ega43.asm écrite pour fichier .COM
;       *******************************************

CODE    SEGMENT

        MOV     AX,ØØØØH       ; Chargement segment
        MOV     DS,AX          ; de code avec ØØØØ.
        MOV     AL,ØØØØØØØ1B   ; Curseur visible en 43 lignes
        OR      DS:Ø487H,AL    ; (forçage bit n°Ø à 1).

        MOV     BL,Ø           ; Sélection bloc n° Ø.
        MOV     AX,1112H       ; Chargement des caractères 8x8
        INT     1ØH            ; (code 12, fonction 11).
        RET                    ; Retour au DOS.

CODE    ENDS
        END
```

Utilisant la fonction 11 de l'interruption 10H du Bios de la carte EGA, la routine nécessite le positionnement préalable du bit n° 0 de l'octet d'adresse 0000:0487. Sans cette précaution, le curseur demeure invisible en mode 43 lignes.

Quant à la fonction elle-même, elle sélectionne le jeu de caractères n° 0 présent en mémoire morte sur la carte EGA (format 8 × 8) de façon à pouvoir afficher 43 lignes de textes à l'écran (8 × 43 = 344).

Création d'un fichier .COM

Destinée à devenir autonome sous la forme d'un fichier .COM, cette routine ne nécessite aucune des habituelles déclarations : pas d'affectation du segment de code, de déclaration de type public et de procédure FAR, avec ce que cela implique de sauvegarde du pointeur de base et du transfert du pointeur de pile en BP.

Ici, tout cela est inutile !... Il suffit de déclarer le segment de code, un point c'est tout !

Quant à la procédure qui permet d'obtenir un fichier .COM à partir du code source de la routine (.ASM), il suffit de lancer comme suit les 3 utilitaires que voici :

```
>MASM EGA43;

>LINK EGA43;

>EXE2BIN EGA43 EGA43.COM
```

Le résultat est un fichier .COM de 19 octets de long qui ne demande qu'à être incorporé à un fichier .BAT.

Important : *Utilisée au sein d'un langage de programmation, cette routine peut ne pas fonctionner correctement si le langage contrecarre son action. C'est notamment le cas du QuickBasic 4.0.*

10

CHRONOMÉTRER AU 1/18e DE SECONDE

Mesurer au 1/18e de seconde le temps d'exécution d'une tâche est l'une des nombreuses possibilités qu'offre cette routine de chronométrage. Idéale pour ceux qui ne disposent que d'un PC/XT sans carte d'horloge.

Routine en Assembleur

```
;          ***************************************************
;          Librairie chrono.asm appelable à partir du Turbo C
;          ***************************************************

CODE        SEGMENT BYTE 'CODE'
            ASSUME  CS:CODE

            PUBLIC  _startchrono
            PUBLIC  _stopchrono

;-----------------------------
_startchrono PROC     FAR

              MOV     CX,0          ; Raz poids faible (par CX)
              MOV     DX,0          ; et poids fort (par DX)
              MOV     AH,1          ; du compteur horaire
              INT     1AH           ; (fonction 1 interruption 1A).
                                    ;
              RET                   ; Retour à l'envoyeur.

_startchrono ENDP
;-----------------------------
```

Routine en Assembleur (suite)

```
_stopchrono  PROC    FAR

             MOV     AH,Ø          ; Lecture du compteur horaire
             INT     1AH           ; (fonction Ø interruption 1A).
             MOV     AX,DX         ; Renvoi poids faible en AX
             MOV     DX,CX         ; et poids fort en DX.
                                   ;
             RET                   ; Retour à l'envoyeur.

_stopchrono  ENDP
;------------------------------
CODE        ENDS
            END
```

Il existe quelque part au sein de la machine un compteur horaire qui est incrémenté 18,2 fois par seconde (à quelques décimales près...). Pourquoi cette valeur bizarre, vous demandez-vous ?... Parce que cela correspond à une fréquence horloge de 1,193180 MHz divisée par 65536 (2^{16}).

Ce compteur horaire peut être remis à zéro grâce à la fonction 1 de l'interruption 1A du Bios, à condition de précharger avant l'appel de cette interruption les registres CX et DX avec des zéros. C'est le rôle dévolu à la routine startchrono de la librairie chrono.

L'arrêt du chronomètre, quant à lui, repose sur la routine stopchrono. Cette dernière fait appel à la fonction 0 de l'interruption 1A du Bios qui charge les registres DX (poids faible) et CX (poids fort) avec les 4 octets du compteur horaire.

Notez que la routine réexpédie le tout par le couple DX:AX vers la fonction appelante et que, pour ce faire, il a fallu quelque peu permuter le contenu des registres...

Programme d'essai en Turbo C

```
main()
{
  int i,temps;

  startchrono();
  for(i=0;i<30000;i++);
  temps=stopchrono();

  printf("Temps écoulé: %d/18e s.",temps);
}
```

Après remise à zéro du compteur avec startchrono, une boucle inutile tournant sur elle-même 30 000 fois simule une opération assez longue dont on veut mesurer le temps d'exécution, ce que fait précisément la fonction stopchrono qui affecte à la variable temps la valeur lue (en 1/18e de seconde) dans le compteur horaire.

Notez que pour les besoins (limités...) de cet essai et bien que la fonction réexpédie un entier long sur 4 octets, nous n'avons jugé utile de n'en prélever que 2 pour l'affichage.

11

LIRE L'HEURE

Pouvoir incruster l'heure dans un masque d'écran ou déclencher une suite d'opérations à une heure donnée, voire les arrêter, sont quelques-unes des applications que vous propose cette routine d'affichage de l'heure. La précision disponible est de 1/100e de seconde. Ne fonctionne que sur AT ou sur PC/XT équipé d'une carte horloge.

Routine en Assembleur

```
;         ************************************************
;         Routine time.asm appelable à partir du Turbo C
;         ************************************************

CODE      SEGMENT BYTE 'CODE'
          ASSUME  CS:CODE
          PUBLIC  _time
_time     PROC    FAR

          PUSH    BP            ; Sauvegarde pointeur de base.
          MOV     BP,SP         ; Transfert pointeur de pile en BP.
          PUSH    SI            ; Sauvegarde index source.
                                ;
          MOV     AH,Ø2CH       ; Lecture de l'heure (résultat en CX DX)
          INT     21H           ; (fonction 2C interruption 21).
          XOR     AH,AH         ; Nettoyage du registre AH.
```

Routine en Assembleur (suite)

```
                                        ;
            MOV     SI,(BP+06H)         ; Chargement adresse variable.
            MOV     AL,CH               ; Transfert des heures en AL
            MOV     [SI],AX             ; et expédition vers le programme.
                                        ;
            MOV     SI,(BP+08H)         ; Chargement adresse variable.
            MOV     AL,CL               ; Transfert des minutes en AL
            MOV     [SI],AX             ; et expédition vers le programme.
                                        ;
            MOV     SI,(BP+0AH)         ; Chargement adresse variable.
            MOV     AL,DH               ; Transfert des secondes en AL
            MOV     [SI],AX             ; et expédition vers le programme.
                                        ;
            MOV     SI,(BP+0CH)         ; Chargement adresse variable.
            MOV     AL,DL               ; Transfert des centièmes en AL
            MOV     [SI],AX             ; et expédition vers le programme.
                                        ;
            POP     SI                  ; Rétablissement index source.
            POP     BP                  ; Rétablissement pointeur de base
            RET                         ; ... et retour à l'envoyeur.

_time       ENDP
CODE        ENDS
            END
```

Mettant à profit la fonction 2C de l'interruption 21 du Bios, cette routine récupère les heures, minutes, secondes et centièmes de seconde de l'horloge interne dans les registres :

CH = Heures
CL = Minutes
DH = Secondes
DL = Centièmes de seconde

Notez que l'on expédie vers le programme le registre AX au complet, ce qui nécessite un nettoyage préalable de la partie haute de ce registre (AH) afin qu'il ne vienne pas perturber la transmission.

Programme d'essai en Turbo C

```
main()
{
  int heure,minute,seconde;
  do
  {
    time(&heure,&minute,&seconde);
    locate(25,1);
    printf("%2d:%2d:%2d",heure,minute,seconde);
  }
  while(heure!=24);
}
```

Ce programme d'essai récupère au travers de la fonction time les paramètres heure, minute et seconde sur les 4 que lui transmet la routine (l'affichage des 1/100e étant d'un intérêt douteux...) afin de les afficher dans le coin inférieur gauche de l'écran (ligne 25, colonne 1).

Notez que l'on emploie la routine locate décrite précédemment pour positionner les chiffres à l'écran. Sans cela, l'affichage défilerait en permanence.

12

LIRE LA DATE

Au même titre que l'heure, il peut être intéressant de disposer de la date au sein de vos programmes, ne serait-ce que pour pouvoir l'incorporer dans un en-tête de fichier par exemple. Attention toutefois, cette routine ne fonctionne que sur AT ou sur PC/XT équipé d'une carte horloge.

Routine en Assembleur

```
;         ********************************************
;         Routine date.asm appelable à partir du Turbo C
;         ********************************************

CODE      SEGMENT BYTE 'CODE'
          ASSUME  CS:CODE
          PUBLIC  _date
_date     PROC    FAR

          PUSH    BP           ; Sauvegarde pointeur de base.
          MOV     BP,SP        ; Transfert pointeur de pile en BP.
          PUSH    SI           ; Sauvegarde index source.
                               ;
          MOV     AH,2AH       ; Lecture de la date
          INT     21H          ; (fonction 2C interruption 21).
          XOR     AH,AH        ; Nettoyage du registre AH.
```

Routine en Assembleur (suite)

```
                                       ;
            MOV     SI,(BP+06H)        ; Adresse variable jour.
            MOV     [SI],AX            ; Expédition vers le programme.
                                       ;
            MOV     SI,(BP+08H)        ; Adresse variable quantième.
            MOV     AL,DL              ; Transfert du quantième en AL
            MOV     [SI],AX            ; et expédition vers le programme.
                                       ;
            MOV     SI,(BP+0AH)        ; Adresse variable mois.
            MOV     AL,DH              ; Transfert du mois en AL
            MOV     [SI],AX            ; et expédition vers le programme.
                                       ;
            MOV     SI,(BP+0CH)        ; Adresse variable année.
            MOV     [SI],CX            ; Expédition vers le programme.
                                       ;
            POP     SI                 ; Rétablissement index source.
            POP     BP                 ; Rétablissement pointeur de base
            RET                        ; ... et retour à l'envoyeur.

_date       ENDP
CODE        ENDS
            END
```

Utilisant la fonction 2A de l'interruption 21 du Bios, cette routine date réexpédie vers le programme appelant le jour, quantième, mois et année de l'horloge interne par le biais des registres :

AL = jour de la semaine (0 = Dimanche, 1 = Lundi, etc...)
DL = Quantième du mois
DH = Mois
CX = Année

Notez le transfert du quantième (dans DL) et du mois (dans DH) dans le registre AL. De cette façon, et sachant que AH est égal à zéro, il n'y a aucun risque de voir l'un de ces 2 paramètres perturber l'autre.

Programme d'essai en Turbo C

```
main()
{
  int jour,quantieme,mois,annee;
  date(&jour,&quantieme,&mois,&annee);

  printf("Date: ");

  if(jour==1) printf("Lundi ");
  if(jour==2) printf("Mardi ");
  if(jour==3) printf("Mercredi ");
  if(jour==4) printf("Jeudi ");
  if(jour==5) printf("Vendredi ");
  if(jour==6) printf("Samedi ");
  if(jour==0) printf("Dimanche ");

  printf("%d/%d/%d",quantieme,mois,annee);
}
```

Après déclaration des variables jour, quantième, mois et année, le programme expédie la fonction date vers la carte horloge pour qu'elle en ramène la valeur de ces 4 paramètres.

Cela étant, il reste à traduire en clair la valeur du paramètre jour (0 = Dimanche, 1 = Lundi, etc...). C'est le rôle dévolu aux 7 lignes de branchement conditionnel qui suivent, la dernière ligne assurant, quant à elle, l'affichage des 3 autres paramètres sous forme de nombres.

13

JOUER UNE NOTE DE MUSIQUE

Jouer une note de musique dont la fréquence et la durée sont paramétrables est ce que vous propose la routine que voici. De surcroît, livrée avec un temps de maintien de 7/8e de la durée totale et un temps de silence de 1/8e, elle assure une bien meilleure articulation des notes lorsque celles-ci sont jouées les unes à la suite des autres.

Routine en Assembleur

```
;           *********************************************
;           Routine sound.asm appelable à partir du Turbo C
;           *********************************************

CODE        SEGMENT WORD 'CODE'
            ASSUME  CS:CODE
            PUBLIC  _sound
_sound      PROC    FAR

            PUSH    BP              ; Sauvegarde pointeur de base.
            MOV     BP,SP           ; Transfert pointeur de pile en BP.
                                    ;
            MOV     AL,ØB6H         ; Code (secret...) d'accès
            OUT     43H,AL          ; au diviseur de fréquence.
                                    ;
            MOV     AX,34DCH        ; Horloge à 1.193180 Mhz chargée en
            MOV     DX,12H          ; AX (poids faible) et DX (poids fort).
            MOV     BX,(BP+Ø6H)     ; Fréquence de la note (en Hertz).
            DIV     BX              ; Calcul coefficient (résultat en AX).
```

Routine en Assembleur (suite)

```
                                  ;
        OUT     42H,AL            ; Transfert coefficient
        MOV     AL,AH             ; au diviseur de fréquence
        OUT     42H,AL            ; (AL d'abord, AH ensuite).
                                  ;
        MOV     BX,(BP+Ø8H)       ; Durée de la note (en 1/256e de seconde).
        MOV     DX,BX             ; Temps de silence en BX
        SHL     DX,1              ; (1/8e de la durée totale).
        SHL     DX,1              ; Calcul du temps de maintien
        SHL     DX,1              ; (7/8e de la durée totale) par
        SUB     DX,BX             ; multiplication par 8 et soustraction.
                                  ;
        IN      AL,61H            ; Lecture port n°61 et
        PUSH    AX                ; sauvegarde sur la pile.
        OR      AL,ØØØØØØ11B      ; Mise en route haut-parleur
        OUT     61H,AL            ; (bits n°Ø et 1 à un).
                                  ;
S1:     MOV     CX,ØAØØH          ; Chargement temps élémentaire en CX
S2:     LOOP    S2                ; (équivalent à 1/256e de seconde).
        DEC     DX                ; Le temps de maintien est-il écoulé ?
        JNZ     S1                ; Non, alors on recommence.
                                  ;
        POP     AX                ; Extinction du haut-parleur par
        OUT     61H,AL            ; rétablissement du port n°61.
                                  ;
S3:     MOV     CX,ØAØØH          ; Chargement temps élémentaire en CX
S4:     LOOP    S4                ; (équivalent à 1/256e de seconde).
        DEC     BX                ; Le temps de silence est-il écoulé ?
        JNZ     S3                ; Non, alors on recommence.
                                  ;
        POP     BP                ; Rétablissement pointeur de base
        RET                       ; ... et retour à l'envoyeur.

_sound  ENDP
CODE    ENDS
        END
```

En quelques mots, sachez qu'il existe quelque part dans la machine un diviseur de fréquence paramétrable à souhait. On y accède à l'aide du code d'accès B6 que l'on expédie sur le port n° 43H.

Sachant qu'il est alimenté par une horloge à 1,193180 MHz et, connaissant la fréquence de la note à produire, on calcule la valeur du cœfficient diviseur par simple division, le résultat étant alors expédié octet par octet vers le port n° 42H.

On calcule ensuite les temps de maintien et de silence de la note qui correspondent respectivement aux 7/8e et 1/8e de la durée totale. Ceci correspond au mode normal d'articulation des notes (cf Graphisme et son sur IBM PC/XT AT et compatibles). D'autres modes peuvent, bien entendu, être envisagés.

Ceci fait, on active le haut-parleur en forçant à un les bits n° 0 et 1 du port 61H pendant un laps de temps multiple de 7 temps élémentaires de 1/256e de seconde.

Notez à ce propos que le temps élémentaire à charger dans CX est dépendant de l'horloge de votre machine, selon le tableau que voici :

Horloge	Valeur(*) à charger dans CX
4,77 MHz	0400
6 MHz	0780
8 MHz	0A00
10 MHz	0C80
12 MHz	0F00

* Exprimée en hexadécimal

Ce temps de maintien étant écoulé, on procède alors à l'extinction du haut-parleur en rétablissant les valeurs d'origine du port 61H et, avant de rendre la main, on exécute le temps de silence de la note.

Programme d'essai en Turbo C

```
main()
{
  sound(850,1);
  sound(700,1);
}
```

Que dire de ce programme si ce n'est que nous nous sommes amusés à reproduire le plus fidèlement possible le double bip d'un certain éditeur pleine page... A vous de deviner lequel !

Quant aux tableaux de codage des notes en fréquence et en durée, les voici :

Notes	Octave n° 1	Octave n° 2	Octave n° 3	Octave n° 4
DO		262	523	1047
DO#		277	554	1109
RE		294	587	1175
RE#		311	622	1245
MI		330	659	1319
FA	175	349	698	
FA#	185	370	740	
SOL	196	392	784	
SOL#	208	415	831	
LA	220	440	880	
LA#	233	466	932	
SI	247	494	988	

Fréquence des notes

Note	Durée
Ronde	64
Blanche	32
Noire	16
Croche	8
Double croche	4
Triple croche	2
Quadruple croche	1

Note	Durée
Blanche pointée	48
Noire pointée	24
Croche pointée	12

Durée des notes

Cela n'est pas sans rappeler les tableaux des instructions SOUND et PLAY des Basics, n'est-ce pas ?... Seule petite différence : le code de durée est inversé par rapport à eux.

14

PASSER EN MODE GRAPHIQUE EGA

Pour peu que vous disposiez d'une version ancienne (de quelques mois...) d'un langage de développement, vous ne pouvez accéder au mode graphique EGA faute d'instruction. Cette lacune est comblée par la routine screen que voici qui, loin de se limiter au seul mode EGA, va vous permettre de sélectionner le mode d'affichage de votre choix (si tant est que la carte graphique le possède...).

Routine en Assembleur

```
;           **************************************************
;           Routine screen.asm appelable à partir du Turbo C
;           **************************************************
;
;
CODE        SEGMENT BYTE 'CODE'
            PUBLIC  _screen
            ASSUME  CS:CODE
_screen     PROC    FAR

            PUSH    BP              ; Sauvegarde pointeur de base.
            MOV     BP,SP           ; Transfert pointeur de pile en BP.
            MOV     AX,(BP+06H)     ; Chargement mode graphique en AX
            INT     10H             ; et appel de l'interruption 10H.
            POP     BP              ; Rétablissement pointeur de base
            RET                     ; ... et retour à l'envoyeur.

_screen     ENDP
CODE        ENDS
            END
```

Que dire sur cette routine ?... Rien de bien particulier, si ce n'est qu'il faut charger le registre AX avec l'argument :

2 pour le mode texte
16 pour le mode graphique EGA

... et expédier le tout à l'interruption 10H du Bios.

A titre d'information, il faut savoir que le Bios est actualisé, tout du moins en ce qui concerne les interruptions graphiques, par la carte graphique implantée dans la machine. Ainsi, il n'y a pas lieu de craindre un non fonctionnement de l'interruption 10H avec une carte graphique récente sous prétexte qu'on dispose d'un Bios ancien.

Programme d'essai en Turbo C

```
main()
{
  screen(16);                    /* passage en mode EGA  */
}
```

Normalement, vous devriez passer en mode EGA, ce qui se traduit sur l'écran par la disparition du curseur clignotant. Pour ceux qui disposent d'un moniteur multisynchrone, les changements de mode sont audibles à l'oreille (un relais de commutation change alors d'état...), permettant ainsi de mieux repérer l'instant de ce changement de mode.

Un dernier point de détail à noter : l'interruption 10H qui assure le passage au mode graphique EGA assure en même temps l'effacement de l'écran (le Bios est ainsi fait...). Il convient donc d'en tenir compte lors du développement d'une application graphique dans laquelle on envisage de faire des appels au mode texte à partir de l'écran graphique.

15

DÉLIMITER UNE FENÊTRE GRAPHIQUE

Délimiter une fenêtre graphique sur l'écran est une nécessité à laquelle il vous faut faire face si vous décidez d'employer quelques-unes des routines graphiques qui suivent, ne serait-ce que pour interdire l'accès à des octets de mémoire qui n'ont rien à voir avec la carte graphique sous prétexte que les pixels à afficher ont des coordonnées débordant les limites de l'écran.

Routine en Assembleur

Que faut-il pour délimiter une fenêtre graphique ?... Bien peu de choses en vérité, puisqu'il suffit de placer dans 4 variables les limites GAUCHE, HAUTE, DROITE et BASSE de la fenêtre, variables que les routines de dessin iront consulter afin de déterminer ce qui doit être dessiné de ce qui ne doit pas l'être.

```
;        ************************************************
;        Librairie ega.asm appelable à partir du Turbo C
;        ************************************************

DATA     SEGMENT WORD 'DATA'

GAUCHE   DW      Ø            ; Limite gauche de la fenêtre.
HAUT     DW      Ø            ; Limite haute de la fenêtre.
DROITE   DW      639          ; Limite droite de la fenêtre.
BAS      DW      349          ; Limite basse de la fenêtre.

DATA     ENDS
DGROUP   GROUP   DATA
```

Routine en Assembleur (suite)

```
CODE        SEGMENT BYTE 'CODE'
            ASSUME  CS:CODE
            ASSUME  DS:DGROUP

            PUBLIC  _screen
            PUBLIC  _viewport
;------------------------------
_screen     PROC    FAR
            ;
_screen     ENDP
;------------------------------
_viewport   PROC    FAR

            PUSH    BP              ; Sauvegarde pointeur de base.
            MOV     BP,SP           ; Transfert pointeur de pile en BP.
                                    ;
            MOV     AX,(BP+Ø6H)     ; Chargement limite gauche.
            MOV     BX,(BP+Ø8H)     ; Chargement limite haute.
            MOV     CX,(BP+ØAH)     ; Chargement limite droite.
            MOV     DX,(BP+ØCH)     ; Chargement limite basse.
                                    ;
            CMP     CX,AX           ; Limite droite > limite gauche ?
            JG      V1              ; Oui, alors c'est tout bon.
            XCHG    CX,AX           ; Non, alors on permute.
                                    ;
V1:         CMP     AX,Ø            ; Limite gauche >= Ø ?
            JGE     V2              ; Oui, alors c'est tout bon.
            MOV     AX,Ø            ; Non, alors limite gauche = Ø.
V2:         CMP     CX,639          ; Limite droite  <= 639 ?
            JLE     V3              ; Oui, alors c'est tout bon.
            MOV     CX,639          ; Non, alors limite droite = 639.
                                    ;
V3:         CMP     DX,BX           ; Limite basse > limite haute ?
            JG      V4              ; Oui, alors c'est tout bon.
            XCHG    DX,BX           ; Non, alors on permute.
                                    ;
V4:         CMP     BX,Ø            ; Limite haute >= Ø ?
            JGE     V5              ; Oui, alors c'est tout bon.
            MOV     BX,Ø            ; Non, alors limite haute = Ø.
V5:         CMP     DX,349          ; Limite basse <= 349 ?
            JLE     V6              ; Oui, alors c'est tout bon.
            MOV     DX,349          ; Non, alors limite basse = 349.
```

Routine en Assembleur (suite)

```
                                  ;
V6:         MOV     GAUCHE,AX     ; Sauvegarde limite gauche.
            MOV     HAUT,BX       ; Sauvegarde limite haute.
            MOV     DROITE,CX     ; Sauvegarde limite droite.
            MOV     BAS,DX        ; Sauvegarde limite basse.
                                  ;
            POP     BP            ; Rétablissement pointeur de base
            RET                   ; ... et retour à l'envoyeur.

_viewport ENDP
;-------------------------------
CODE        ENDS
            END
```

Le seul point important de cette routine concerne la mise en commun des variables du segment de données, en regroupant les routines au sein d'une même librairie. Ainsi, ces variables pourront être consultées à partir des autres routines.

Cette façon de procéder est fortement recommandée chaque fois que plusieurs routines doivent accéder à des variables communes. D'autres méthodes existent bien entendu, mais celle-ci est la plus facile à mettre en œuvre.

Quant à la routine elle-même, elle assure le chargement dans les registres AX, BX, CX et DX des 4 limites de la fenêtre, les compare deux à deux ainsi qu'aux limites absolues de façon à vérifier leur validité, voire les permute au besoin, et termine en transférant le contenu des 4 registres dans les variables communes.

Programme d'essai en Turbo C

```
main()
{
  screen(16);
  viewport(Ø,Ø,639,349);              /* Délimitation fenêtre */
}
```

Bien que sans effet (tout du moins à ce stade...), ce petit programme d'essai de la routine viewport permet de vérifier que le passage des limites de la fenêtre s'effectue sans plantage.

16

TIRER UN TRAIT EN XOR

Dessiner un trait en mode XOR, c'est-à-dire sans altérer le fond d'image, fait partie de ces choses que les langages actuels ne savent pas faire (en tout cas pas facilement...). C'est pour pallier à cette insuffisance que nous vous proposons la routine xline que voici. Bien entendu, elle sait aussi dessiner en mode normal. Quant à sa vitesse d'affichage des pixels, sachez qu'elle est (à l'heure actuelle...) plus rapide que la plus rapide des routines de dessin de traits que proposent les langages disposant d'une librairie graphique (Basic, QuickBasic, TurboBasic, TurboPascal, C, Quick C et Turbo C).

Routine en Assembleur

```
;           ******************************************************
;           Librairie ega.asm appelable à partir du Turbo C
;           ******************************************************

DATA        SEGMENT WORD 'DATA'

GAUCHE      DW      Ø               ; Limite gauche de la fenêtre.
HAUT        DW      Ø               ; Limite haute de la fenêtre.
DROITE      DW      639             ; Limite droite de la fenêtre.
BAS         DW      349             ; Limite basse de la fenêtre.
DELTAX      DW      ?               ; Deltax.
DELTAY      DW      ?               ; Deltay.
FLAG        DW      ?               ; Drapeau à tout faire.
XD          DW      ?               ; Sauvegarde de Xd.
XF          DW      ?               ; Sauvegarde de Xf.
YD          DW      ?               ; Sauvegarde de Yd.
YF          DW      ?               ; Sauvegarde de Yf.
```

Routine en Assembleur (suite)

```
DATA      ENDS
DGROUP    GROUP   DATA

CODE      SEGMENT BYTE 'CODE'
          ASSUME  CS:CODE
          ASSUME  DS:DGROUP

          PUBLIC  _screen
          PUBLIC  _viewport
          PUBLIC  _xline

;-----------------------------
_screen   PROC    FAR
          ;;
_screen   ENDP
;-----------------------------
_viewport PROC    FAR
          ;;
_viewport ENDP
;-----------------------------
_xline    PROC    FAR

          PUSH    BP           ; Sauvegarde pointeur de base.
          MOV     BP,SP        ; Transfert pointeur de pile en BP.
          PUSH    ES           ; Sauvegarde extra segment.
          PUSH    SI           ; Sauvegarde index source.
          PUSH    DI           ; Sauvegarde index destination.
                               ;
          MOV     DX,0A000H    ; Chargement adresse de début
          MOV     ES,DX        ; dans extra segment via DX.
          MOV     DX,03CEH     ; Port du contrôleur graphique.
          MOV     AX,0205H     ; Ecriture sur les 4 plans EGA
          OUT     DX,AX        ; (code 2 registre 5).
                               ;
          MOV     BX,(BP+0EH)  ; Transfert couleur en BX.
          CMP     BX,128       ; XOR or not XOR ?
          JLE     L1           ; Non, alors c'est tout bon.
          MOV     AX,1803H     ; Passage en mode XOR
          OUT     DX,AX        ; (code 18 registre 3).
                               ;
L1:       MOV     SI,(BP+0CH)  ; Transfert coordonnée Yf en SI.
          MOV     DI,(BP+0AH)  ; Transfert coordonnée Xf en DI.
          MOV     AX,(BP+08H)  ; Transfert coordonnée Yd en AX.
          MOV     CX,(BP+06H)  ; Transfert coordonnée Xd en CX.
```

Routine en Assembleur (suite)

```
                                  ;
         CMP     DI,CX            ; Xf > Xd ?
         JG      L3               ; Oui, alors on laisse en l'état.
         JL      L2               ; Non, alors on permute.
         CMP     SI,AX            ; Egal, dans ce cas Yf >= Yd ?
         JGE     L3               ; Oui, alors on laisse en l'état.
L2:      XCHG    DI,CX            ; Non, alors on permute Xf avec Xd
         XCHG    SI,AX            ; ainsi que Yf avec Yd.
                                  ;
L3:      CMP     CX,GAUCHE        ; Xd >= limite gauche ?
         JGE     L4               ; Oui, alors test suivant.
         CMP     DI,GAUCHE        ; Xf lui aussi < limite gauche ?
         JL      L7               ; Oui, alors trait invisible.
                                  ;
L4:      CMP     DI,DROITE        ; Xf <= limite droite ?
         JLE     L5               ; Oui, alors test suivant.
         CMP     CX,DROITE        ; Xd lui aussi > limite droite ?
         JG      L7               ; Oui, alors trait invisible.
                                  ;
L5:      CMP     AX,HAUT          ; Yd >= limite haute ?
         JGE     L6               ; Oui, alors test suivant.
         CMP     SI,HAUT          ; Yf lui aussi < limite haute ?
         JL      L7               ; Oui, alors trait invisible.
                                  ;
L6:      CMP     SI,BAS           ; Yf <= limite basse ?
         JLE     L8               ; Oui, alors rubrique suivante.
         CMP     AX,BAS           ; Yd lui aussi <= limite basse ?
         JLE     L8               ; Oui, alors rubrique suivante.
L7:      JMP     L42              ; Non, alors trait invisible.
                                  ;
L8:      CMP     SI,AX            ; Delta y = Ø ?
         JNE     L11              ; Non, alors test suivant.
         CMP     CX,GAUCHE        ; Xd >= limite gauche ?
         JGE     L9               ; Oui, alors test suivant.
         MOV     CX,GAUCHE        ; Non, alors Xd = limite gauche.
L9:      CMP     DI,DROITE        ; Xf <= limite droite ?
         JLE     L1Ø              ; Oui, alors on dessine.
         MOV     DI,DROITE        ; Non, alors Xf = limite droite.
L1Ø:     JMP     L25              ; Vers dessin des traits.
                                  ;
L11:     CMP     DI,CX            ; Delta x = Ø ?
         JNE     L14              ; Non, alors trait oblique.
         CMP     AX,HAUT          ; Yd >= limite haute ?
         JGE     L12              ; Oui, alors test suivant.
         MOV     AX,HAUT          ; Non, alors Yd = limite haute.
L12:     CMP     SI,BAS           ; Yf <= limite basse ?
         JLE     L13              ; Oui, alors on dessine.
         MOV     SI,BAS           ; Non, alors Yf = limite basse.
L13:     JMP     L25              ; Vers dessin des traits.
```

Routine en Assembleur (suite)

```
                                  ;
L14:      MOV     XD,CX           ; Sauvegarde de Xd.
          MOV     XF,DI           ; Sauvegarde de Xf.
          MOV     YD,AX           ; Sauvegarde de Yd.
          MOV     YF,SI           ; Sauvegarde de Yf.
          SUB     DI,CX           ; Calcul (provisoire) de delta x.
          SUB     SI,AX           ; Calcul (provisoire) de delta y.
                                  ;
          MOV     CX,GAUCHE       ; Xlim = limite gauche.
          MOV     AX,XD           ; Transfert de Xd en AX.
          SUB     AX,CX           ; Xd - limite gauche >= Ø ?
          JGE     L17             ; Oui, alors test suivant.
          NEG     AX              ; Yd' = (Xlim - Xd).
          IMUL    SI              ; Yd' = (Xlim - Xd) * SI.
          IDIV    DI              ; Yd' = (Xlim - Xd) * SI / DI.
          ADD     AX,YD           ; Yd' = Yd + (Xlim - Xd) * SI / DI.
          MOV     DX,HAUT         ; Transfert limite haute en DX.
          CMP     AX,DX           ; Yd' >= limite haute ?
          JGE     L15             ; Oui, alors test suivant.
          CMP     YD,DX           ; Yd lui aussi < limite haute ?
          JL      L17             ; Oui, alors voir calcul de Xd'.
          JMP     L42             ; Non, alors trait invisible.
L15:      MOV     DX,BAS          ; Transfert limite basse en DX.
          CMP     AX,DX           ; Yd' <= limite basse ?
          JLE     L16             ; Oui, alors le calcul est bon.
          CMP     YD,DX           ; Yd lui aussi > limite basse ?
          JG      L17             ; Oui, alors voir calcul de Xd'.
          JMP     L42             ; Non, alors trait invisible.
L16:      MOV     YD,AX           ; Yd = Yd'.
          MOV     XD,CX           ; Xd = Ylim (limite gauche).
                                  ;
L17:      MOV     CX,HAUT         ; Ylim  = limite haute.
          MOV     AX,YD           ; Transfert de Yd en AX.
          SUB     AX,CX           ; Yd - Ylim (limite haute) < Ø ?
          JL      L18             ; Oui, alors on calcule Xd'.
          MOV     CX,BAS          ; Ylim = limite basse.
          MOV     AX,YD           ; On recharge Yd en AX.
          SUB     AX,CX           ; Yd - Ylim (limite basse) <= Ø ?
          JLE     L19             ; Oui, alors test suivant.
L18:      NEG     AX              ; Xd' = (Ylim - Yd).
          IMUL    DI              ; Xd' = (Ylim - Yd) * DI.
          IDIV    SI              ; Xd' = (Ylim - Yd) * DI / SI.
          ADD     AX,XD           ; Xd' = Xd + (Ylim - Yd) * DI / SI.
          CMP     AX,GAUCHE       ; Xd' < limite gauche ?
          JL      RELAIS          ; Oui, alors trait invisible.
          CMP     AX,DROITE       ; Xd' > limite droite ?
          JG      RELAIS          ; Oui, alors trait invisible.
          MOV     XD,AX           ; Xd = Xd'.
          MOV     YD,CX           ; Yd = Ylim (limite basse ou haute).
```

Routine en Assembleur (suite)

```
                                   ;
L19:      MOV     CX,DROITE        ; Xlim = limite droite.
          MOV     AX,XF            ; Transfert de Xf en AX.
          SUB     AX,CX            ; Xf - Xlim (limite droite) <= Ø ?
          JLE     L22              ; Oui, alors test suivant.
          NEG     AX               ; Yf' = (Xlim - Xf).
          IMUL    SI               ; Yf' = (Xlim - Xf) * SI.
          IDIV    DI               ; Yf' = (Xlim - Xf) * SI / DI.
          ADD     AX,YF            ; Yf' = Yf + (Xlim - Xf) * SI / DI.
          MOV     DX,HAUT          ; Transfert limite haute en DX.
          CMP     AX,DX            ; Yf' >= limite haute ?
          JGE     L2Ø              ; Oui, alors test suivant.
          CMP     YF,DX            ; Yf lui aussi < limite haute ?
          JL      L22              ; Oui, alors on calcule Xf'.
          JMP     L42              ; Non, alors trait invisible.
L2Ø:      MOV     DX,BAS           ; Transfert limite basse en DX.
          CMP     AX,DX            ; Yf' <= limite basse ?
          JLE     L21              ; Oui, alors le calcul est bon.
          CMP     YF,DX            ; Yf lui aussi > limite basse ?
          JG      L22              ; Oui, alors on calcule Xf'.
RELAIS:   JMP     L42              ; Non, alors trait invisible.
L21:      MOV     YF,AX            ; Yf = Yf'.
          MOV     XF,CX            ; Xf = Xlim (limite droite).
                                   ;
L22:      MOV     CX,BAS           ; Ylim = limite basse.
          MOV     AX,YF            ; Transfert de Yf en AX.
          SUB     AX,CX            ; Yf - Ylim (limite basse) > Ø ?
          JG      L23              ; Oui, alors on calcule Xf.
          MOV     CX,HAUT          ; Ylim = limite haute.
          MOV     AX,YF            ; On recharge Yf en AX.
          SUB     AX,CX            ; Yf - Ylim (limite haute) >= Ø ?
          JGE     L24              ; Oui, alors fin des tests.
L23:      NEG     AX               ; Xf' = (Ylim - Yf).
          IMUL    DI               ; Xf' = (Ylim - Yf) * DI.
          IDIV    SI               ; Xf' = (Ylim - Yf) * DI / SI.
          ADD     AX,XF            ; Xf' = Xf + (Ylim - Yf) * DI / SI.
          CMP     AX,GAUCHE        ; Xf' < limite gauche ?
          JL      RELAIS           ; Oui, alors trait invisible.
          CMP     AX,DROITE        ; Xf' > limite droite ?
          JG      RELAIS           ; Oui, alors trait invisible.
          MOV     XF,AX            ; Xf = Xf'.
          MOV     YF,CX            ; Yf = Ylim (limite basse ou haute).
                                   ;
L24:      MOV     DX,Ø3CEH         ; Rechargement adresse contrôleur.
          MOV     CX,XD            ; Rechargement de Xd en CX.
          MOV     DI,XF            ; Rechargement de Xf en DI.
          MOV     AX,YD            ; Rechargement de Yd en AX.
          MOV     SI,YF            ; Rechargement de Yf en SI.
```

Routine en Assembleur (suite)

```
                                        ;
L25:      SUB     DI,CX                 ; Calcul (définitif) de delta x.
          CMP     SI,AX                 ; Yf >= Yd ?
          JGE     L26                   ; Oui, alors c'est tout bon.
          CMP     DI,Ø                  ; Au fait, est-ce un trait vertical ?
          JNE     L26                   ; Non, alors on laisse en l'état.
          XCHG    SI,AX                 ; Oui, alors on permute Yf avec Yd.
L26:      SUB     SI,AX                 ; Calcul (définitif) de delta y.
          MOV     FLAG,8Ø               ; Mise à +8Ø du drapeau FLAG.
          JGE     L26bis                ; Delta y >= Ø ?
          NEG     SI                    ; Non, alors on le rend positif
          NEG     FLAG                  ; et on met FLAG à -8Ø.
L26bis:   MOV     DELTAX,DI             ; Sauvegarde de delta x.
          MOV     DELTAY,SI             ; Sauvegarde de delta y.
                                        ;
          SHL     AX,1                  ; Calcul de l'adresse de l'octet.
          SHL     AX,1                  ; Multiplication par 8Ø de Yd.
          SHL     AX,1                  ; 1re multiplication par 16
          SHL     AX,1                  ; obtenue en décalant AX de 4 bits.
          MOV     BP,AX                 ; Stockage du résultat en BP.
          SHL     AX,1                  ; 2e multiplication par 64 obtenue
          SHL     AX,1                  ; en décalant AX de 2 bits de plus.
          ADD     BP,AX                 ; Addition du résultat à BP.
          MOV     AX,CX                 ; Transfert de XD en AX.
          SHR     AX,1                  ; Division par 8 de Xd
          SHR     AX,1                  ; réalisée à l'aide de
          SHR     AX,1                  ; 3 décalages vers la droite.
          ADD     BP,AX                 ; Addition du résultat à BP.
          AND     CX,7                  ; CL contient le reste.
                                        ;
          MOV     AX,DI                 ; Delta x et
          OR      AX,SI                 ; delta y = Ø ?
          JZ      L27                   ; Oui, alors c'est un point isolé.
          CMP     DI,Ø                  ; Delta x = Ø ?
          JE      L29                   ; Oui, alors c'est une verticale.
          CMP     SI,Ø                  ; Delta y = Ø ?
          JE      L31                   ; Oui, alors c'est une horizontale.
          JNE     L35                   ; Non, c'est une droite quelconque.
                                        ;
L27:      MOV     AX,8ØØ8H              ; Chargement du masque (8Ø)
          SHR     AH,CL                 ; et son décalage (via CL).
          OUT     DX,AX                 ; Registre 8 port 3CE.
          MOV     BH,ES:[BP]            ; Fausse lecture pour vider la mémoire
          MOV     ES:[BP],BL            ; et allumage du pixel.
L28:      JMP     L42                   ; Le trait est fini.
                                        ;
L29:      MOV     AX,8ØØ8H              ; Chargement du masque (8Ø)
          SHR     AH,CL                 ; et son décalage (via CL).
          OUT     DX,AX                 ; Registre 8 port 3CE.
```

Routine en Assembleur (suite)

```
                                   ;
L30:    MOV     BH,ES:[BP]         ; Fausse lecture pour vider la mémoire
        MOV     ES:[BP],BL         ; et allumage du pixel.
        ADD     BP,80              ; On rajoute 80 à BP.
        DEC     SI                 ; On décrémente delta y.
        JGE     L30                ; Si delta y >= 0 on recommence.
        JMP     L42                ; Le trait est fini.
                                   ;
L31:    MOV     AX,0FF08H          ; Chargement du masque en AX
        SHR     AH,CL              ; et son décalage via CL.
        XOR     CL,00000111B       ; Calcul du complément du reste.
        SUB     CX,DI              ; CX < delta x ?
        JLE     L32                ; Oui, alors le masque est bon.
        SHR     AH,CL              ; Sinon, mise en forme du masque
        SHL     AH,CL              ; grâce à un droite-gauche.
                                   ;
L32:    NEG     CX                 ; CX change de signe.
        OUT     DX,AX              ; Transfert du masque.
        MOV     BH,ES:[BP]         ; Fausse lecture pour vider la mémoire.
        MOV     ES:[BP],BL         ; Allumage du pixel.
        MOV     AH,11111111B       ; On recharge le masque.
                                   ;
L33:    CMP     CX,0               ; CX < 0 ?
        JL      L28                ; Oui, alors le trait est fini.
        CMP     CX,8               ; CX >= 8 ?
        JGE     L34                ; Oui, alors le masque est bon.
        SHR     AH,CL              ; Sinon, on décale le masque
        XOR     AH,11111111B       ; et on le complémente.
                                   ;
L34:    OUT     DX,AX              ; Transfert du masque.
        INC     BP                 ; On incrémente BP.
        MOV     BH,ES:[BP]         ; Fausse lecture pour vider la mémoire.
        MOV     ES:[BP],BL         ; Allumage du pixel.
        SUB     CX,8               ; CX = CX - 8.
        JMP     SHORT L33          ; ...et on recommence.
                                   ;
L35:    MOV     AX,8008H           ; Chargement du masque en AX
        SHR     AH,CL              ; et son positionnement via CL.
        CMP     DI,SI              ; Delta x < delta y ?
        JL      L39                ; Oui, alors c'est pas ici.
        MOV     CX,DI              ; CX = delta x.
        NEG     CX                 ; CX = - delta x.
        SHL     DI,1               ; DI = 2 delta x.
        SHL     SI,1               ; SI = 2 delta y.
```

Routine en Assembleur (suite et fin)

```
                                 ;
L36:      OUT      DX,AX         ; Transfert du masque.
          MOV      BH,ES:[BP]    ; Fausse lecture pour vider la mémoire.
          MOV      ES:[BP],BL    ; Allumage du pixel.
          ADD      CX,SI         ; On rajoute 2 delta y à CX.
          JL       L37           ; Si CX < Ø, c'est tout bon.
          SUB      CX,DI         ; Sinon, on soustrait 2 delta x à CX
          ADD      BP,FLAG       ; et on rajoute +/-8Ø à BP.
L37:      ROR      AH,1          ; On décale le masque.
          JNC      L38           ; A-t-on fait un tour complet ?
          INC      BP            ; Si oui, on incrémente BP.
L38:      DEC      DELTAX        ; On décrémente delta x.
          JGE      L36           ; Si delta x >= Ø on recommence.
          JMP      L42           ; Sinon, le trait est fini.
                                 ;
L39:      MOV      CX,SI         ; CX = delta y.
          NEG      CX            ; CX = - delta y.
          SHL      DI,1          ; DI = 2 delta x.
          SHL      SI,1          ; SI = 2 delta y.
                                 ;
L4Ø:      OUT      DX,AX         ; Transfert du masque.
          MOV      BH,ES:[BP]    ; Fausse lecture pour vider la mémoire.
          MOV      ES:[BP],BL    ; Allumage du pixel.
          ADD      CX,DI         ; On rajoute 2 delta x à CX.
          JL       L41           ; Si CX < Ø, c'est tout bon.
          SUB      CX,SI         ; Sinon, on soustrait 2 delta y à CX
          ROR      AH,1          ; et on décale le masque.
          JNC      L41           ; A-t-on fait un tour complet ?
          INC      BP            ; Si oui, on incrémente BP.
L41:      ADD      BP,FLAG       ; On rajoute +/-8Ø à BP
          DEC      DELTAY        ; et on décrémente delta y.
          JGE      L4Ø           ; Si delta y >= Ø on recommence.
                                 ;
L42:      MOV      DX,Ø3CEH      ; Rechargement adresse contrôleur.
          MOV      AX,ØFFØ8H     ; Désactivation du masque
          OUT      DX,AX         ; (code FF registre 8).
          MOV      AX,ØØØ5H      ; Mode normal d'écriture
          OUT      DX,AX         ; (code Ø registre 5).
          MOV      AX,ØØØ3H      ; Suppression du mode XOR
          OUT      DX,AX         ; (code Ø registre 3).
                                 ;
          POP      DI            ; Rétablissement index destination.
          POP      SI            ; Rétablissement index source.
          POP      ES            ; Rétablissement extra segment.
          POP      BP            ; Rétablissement pointeur de base
          RET                    ; ... et retour à l'envoyeur.

_xline    ENDP
;------------------------------
CODE      ENDS
          END
```

Il serait trop long de développer dans le détail le principe de fonctionnement de cette routine de dessin de trait. Sachez seulement qu'elle s'appuie sur l'algorithme de Bresenham pour ce qui a trait au calcul des coordonnées des pixels et à la bonne vieille méthode de calcul d'un point d'une droite, connaissant un autre point et l'équation de la droite, pour le détourage des traits.

Comme vous pouvez le constater, on commence par charger l'adresse de début de la page graphique EGA dans l'extra segment, via un registre quelconque (DX en l'occurrence...). Ceci fait, on charge une fois pour toute le registre DX avec l'adresse du port du contrôleur graphique (3CE) et on y transfère dans son registre 5 l'ordre de passer en mode d'écriture sur les 4 plans.

Le passage des paramètres étant tout à fait classique, hormis le choix du mode XOR sélectionné en augmentant le numéro de la couleur de 128, on passe directement au calcul de l'adresse de l'octet. Cette dernière s'obtient en multipliant la coordonnée Y par 80 (il y a 80 octets par ligne, c'est-à-dire autant que de caractères...) et en y rajoutant la valeur entière de la coordonnée X divisée par 8, le reste de la division fournissant la position du pixel dans l'octet.

Vous devez peut-être vous demander pourquoi avoir choisi une solution aussi tordue pour effectuer la multiplication par 80, alors qu'il existe dans l'Assembleur des 8088/8086/80286 une instruction MUL ? C'est uniquement pour grignoter quelques microsecondes, l'emploi de l'instruction MUL étant plus pénalisante en cycles machine que l'astuce qui consiste à décomposer la multiplication par 80 en une somme de 2 multiplications multiples d'une puissance de 2 (80 = 16 + 64) réalisables par une suite de 4 et de 6 décalages à gauche.

L'adresse de l'octet étant placée en BP (ce pointeur étant redevenu libre après le chargement des paramètres...), on charge alors en AH un masque correspondant à la position n° 7 dans l'octet (80 = 10000000B) et on le positionne correctement grâce au reste de la division de X par 8 figurant dans CL avant de l'expédier au contrôleur graphique.

Cela fait, on est alors prêt à effectuer la fausse lecture de la mémoire, (le registre BH qui est libre va nous servir à cela...) de façon à être autorisé à écrire les 4 bits de la couleur aux mêmes emplacements de mémoire (la couleur étant en BL...).

Si maintenant on examine de près la première portion de la routine de détourage (il y en a 4 au total, une par limite franchie...), on retrouve, après un test de franchissement de la limite gauche, le calcul de la coordonnée Yd'.

Ce calcul effectué, on vérifie si le résultat obtenue est compris entre les limites haute et basse de la fenêtre. Si ce n'est pas le cas, on le refuse et, après un second test visant à déterminer si le trait est partiellement visible ou non, on procède à un second calcul, mais en considérant cette fois-ci qu'il s'agit du franchissement de la limite haute ou basse (calcul de Xd').

Dans l'éventualité ou le second calcul produirait de nouveau un résultat situé en dehors des limites gauche et droite de la fenêtre, on refuse ce second calcul et on décrète le trait invisible.

Dernier point de détail : le contenu des registres étant détruit par les opérations de multiplication et de division (à l'exception toutefois du registre BX...), il s'en suit qu'il a fallu sauvegarder leur contenu dans 4 nouvelles variables (XD, XF, YD et YF) avant de procéder aux opérations de détourage et ce, de façon à pouvoir rétablir leur contenu avant d'entrer dans la routine de dessin.

Si enfin on s'intéresse à la première des deux sous-routines de dessin des traits obliques (celle qui correspond à la pente < 1), on constate qu'elle débute par le calcul de la relation de Bresenham qu'elle place en CX, les registres DI et SI contenant respectivement l'incrément (2.deltay) et le décrément (2.deltax) dont elle aura besoin par la suite.

Cela étant et après chargement du masque et allumage d'un pixel, on rajoute 2.deltay à CX comme le veut la théorie. Notez que selon le signe de ce dernier, on sait alors s'il convient ou non de soustraire 2.deltax du contenu de CX. Notez également que l'on décale systématiquement d'un bit le masque de positionnement et que l'on passe à la ligne immédiatement supérieure ou inférieure (selon la valeur du drapeau FLAG...) dans le cas où le contenu de CX est positif.

Quant à la seconde sous-routine c'est, à une variante près, la même chose si l'on considère qu'il a fallu permuter deltax avec deltay. La seule différence réside dans le fait qu'il faut passer systématiquement à la ligne immédiatement supérieure ou inférieure après chaque dessin d'un pixel et qu'il ne faut décaler le masque de positionnement que dans le cas où le test de CX révèle un contenu positif.

Programme d'essai en Turbo C

```
int x,y;
main()
{
  screen(16);

  xline(158,98,482,98,14);       /* Cadre autour de la fenêtre */
  xline(482,98,482,252,14);
  xline(158,252,482,252,14);
  xline(158,98,158,252,14);

  viewport(160,100,480,250);     /* Délimitation d'une fenêtre */

  for(y=0;y<350;y++)             /* Traits horizontaux */
    xline(0,y,639,y,1);

  for(x=0;x<640;x++)             /* Traits verticaux */
    xline(x,0,x,349,2);

  for(y=0;y<350;y++)             /* Traits obliques en mode XOR */
    xline(0,0,639,y,131);
  for(x=639;x>-1;x--)
    xline(0,0,x,349,131);
}
```

Reste à vérifier que tout cela fonctionne !... Pour ce faire, essayez donc le petit programme que voici qui dessine, à l'intérieur d'une fenêtre délimitée par viewport, une succession de traits horizontaux, verticaux et obliques.

Notez le résultat obtenu avec la couleur 131 des traits obliques. Elle correspond au dessin en mode XOR (couleur 3 + 128) et donne cet aspect marbré qui laisse apparaître le fond vert précédent chaque fois qu'on dessine deux fois au même emplacement.

Quant à sa vitesse d'exécution, sortez votre chronomètre et comparez...

17

DÉPLACER UN RÉTICULE

Comportant deux traits, un vertical et l'autre horizontal, un réticule permet un pointage de précision sur une image, de meilleure qualité que celui que procure une flèche ou une cible. Or, paradoxe, aucun des langages actuels ne vous permet de réaliser un tel réticule car, ne l'oublions pas, ce dernier doit pouvoir se déplacer sur le fond d'image sans l'altérer. Cette lacune est désormais comblée avec la routine retic que voici.

Routine en Assembleur

```
;          ***********************************************
;          Librairie ega.asm appelable à partir du Turbo C
;          ***********************************************

DATA       SEGMENT WORD 'DATA'

GAUCHE     DW      Ø              ; Limite gauche de la fenêtre.
HAUT       DW      Ø              ; Limite haute de la fenêtre.
DROITE     DW      639            ; Limite droite de la fenêtre.
BAS        DW      349            ; Limite basse de la fenêtre.
DELTAX     DW      ?              ; Deltax.
DELTAY     DW      ?              ; Deltay.
FLAG       DW      ?              ; Drapeau à tout faire.
XD         DW      ?              ; Sauvegarde de Xd.
XF         DW      ?              ; Sauvegarde de Xf.
YD         DW      ?              ; Sauvegarde de Yd.
YF         DW      ?              ; Sauvegarde de Yf.

DATA       ENDS
DGROUP     GROUP   DATA
```

Routine en Assembleur (suite)

```
CODE        SEGMENT BYTE 'CODE'
            ASSUME  CS:CODE
            ASSUME  DS:DGROUP

            PUBLIC  _screen
            PUBLIC  _viewport
            PUBLIC  _xline
            PUBLIC  _retic

;-------------------------------
_screen     PROC    FAR
             ;;
_screen     ENDP
;-------------------------------
_viewport   PROC    FAR
             ;;
_viewport   ENDP
;-------------------------------
_xline      PROC    FAR
             ;;
_xline      ENDP
;-------------------------------
_retic      PROC    FAR

            PUSH    BP            ; Sauvegarde pointeur de base.
            MOV     BP,SP         ; Transfert pointeur de pile en BP.
            PUSH    ES            ; Sauvegarde extra segment.
            PUSH    SI            ; Sauvegarde index source.
            PUSH    DI            ; Sauvegarde index destination.
                                  ;
            MOV     DX,0A000H     ; Chargement adresse de début
            MOV     ES,DX         ; dans extra segment (via DX).
            MOV     DX,03CEH      ; Port du contrôleur graphique.
            MOV     AX,1803H      ; Passage en mode XOR
            OUT     DX,AX         ; (code 18 registre 3).
            MOV     AX,0205H      ; Ecriture sur les 4 plans EGA
            OUT     DX,AX         ; (mode 2 registre 5).
                                  ;
            MOV     CX,(BP+06H)   ; Transfert coordonnée Xr en CX.
            MOV     AX,(BP+08H)   ; Transfert coordonnée Yr en AX.
            MOV     YD,AX         ; Sauvegarde de Yr.
            MOV     BX,(BP+0AH)   ; Transfert couleur en BX.
                                  ;
            CMP     AX,BAS        ; Yr <= limite basse ?
            JLE     R1            ; Oui, alors c'est tout bon.
            JMP     R9            ; Non, alors réticule invisible.
```

Routine en Assembleur (suite)

```
                                        ;
R1:         CMP     AX,HAUT             ; Yr >= limite haute ?
            JGE     R2                  ; Oui, alors c'est tout bon.
            JMP     R9                  ; Non, alors réticule invisible.
                                        ;
R2:         CMP     CX,DROITE           ; Xr <= limite droite ?
            JLE     R3                  ; Oui, alors c'est tout bon.
            JMP     R9                  ; Non, alors réticule invisible.
                                        ;
R3:         CMP     CX,GAUCHE           ; Xr >= limite gauche ?
            JGE     R4                  ; Oui, alors c'est tout bon.
            JMP     R9                  ; Non, alors réticule invisible.
                                        ;
R4:         MOV     SI,BAS              ; Calcul de delta y
            SUB     SI,HAUT             ; (limite basse - limite haute).
            MOV     DI,DROITE           ; Calcul de delta x
            SUB     DI,GAUCHE           ; (limite droite - limite gauche).
                                        ;
            MOV     AX,HAUT             ; Haut du trait vertical en AX.
            SHL     AX,1                ; Calcul de l'adresse de l'octet.
            SHL     AX,1                ; Multiplication par 80.
            SHL     AX,1                ; 1re multiplication par 16
            SHL     AX,1                ; obtenue en décalant AX de 4 bits.
            MOV     BP,AX               ; Stockage du résultat en BP.
            SHL     AX,1                ; 2e multiplication par 64 obtenue
            SHL     AX,1                ; en décalant AX de 2 bits de plus.
            ADD     BP,AX               ; Addition du résultat à BP.
            MOV     AX,CX               ; Transfert de Xr en AX.
            SHR     AX,1                ; Division par 8 de Xr
            SHR     AX,1                ; réalisée en décalant AX
            SHR     AX,1                ; de 3 bits vers la droite.
            ADD     BP,AX               ; Adresse des 4 octets en BP.
            AND     CX,7                ; Position dans octets en CL.
                                        ;
            MOV     AX,8008H            ; Chargement du masque
            SHR     AH,CL               ; et son décalage via CL.
            OUT     DX,AX               ; Transfert au contrôleur.
R5:         MOV     BH,ES:[BP]          ; Fausse lecture pour vider la mémoire
            MOV     ES:[BP],BL          ; et allumage du pixel.
            ADD     BP,80               ; DI = DI + 80.
            DEC     SI                  ; Delta y - 1 >= 0 ?
            JGE     R5                  ; Oui, alors on recommence.
```

Routine en Assembleur (suite)

```
                                   ;
         MOV     AX,YD             ; Transfert de Yr en AX.
         MOV     CX,GAUCHE         ; Gauche du trait horizontal en CX.
         SHL     AX,1              ; Calcul de l'adresse de l'octet.
         SHL     AX,1              ; Multiplication par 8Ø de Yr.
         SHL     AX,1              ; 1re multiplication par 16
         SHL     AX,1              ; obtenue en décalant AX de 4 bits.
         MOV     BP,AX             ; Stockage du résultat en BP.
         SHL     AX,1              ; 2e multiplication par 64 obtenue
         SHL     AX,1              ; en décalant AX de 2 bits de plus.
         ADD     BP,AX             ; Addition du résultat à BP.
         MOV     AX,CX             ; Transfert de CX en AX.
         SHR     AX,1              ; Division par 8
         SHR     AX,1              ; réalisée en décalant AX
         SHR     AX,1              ; de 3 bits vers la droite.
         ADD     BP,AX             ; Adresse des 4 octets en BP.
         AND     CX,7              ; Position dans octets en CL.
                                   ;
         MOV     AX,ØFFØ8H         ; Chargement du masque en AX
         SHR     AH,CL             ; et son décalage via CL.
         XOR     CL,7              ; Calcul du complément du reste.
         SUB     CX,DI             ; CX < delta x ?
         JLE     R6                ; Oui, alors le masque est bon.
         SHR     AH,CL             ; Mise en forme du masque
         SHL     AH,CL             ; grâce à un droite-gauche.
R6:      NEG     CX                ; CX change de signe.
         OUT     DX,AX             ; Transfert au contrôleur.
         MOV     BH,ES:[BP]        ; Fausse lecture et
         MOV     ES:[BP],BL        ; allumage du pixel.
         MOV     AH,11111111B      ; On recharge le masque.
                                   ;
R7:      CMP     CX,Ø              ; CX < Ø ?
         JL      R9                ; Oui, alors c'est fini.
         CMP     CX,8              ; CX >= 8 ?
         JGE     R8                ; Oui, alors le masque est bon.
         SHR     AH,CL             ; Sinon, on décale le masque
         XOR     AH,11111111B      ; et on le complémente.
R8:      OUT     DX,AX             ; Transfert du masque.
         INC     BP                ; BP = BP + 1.
         MOV     BH,ES:[BP]        ; Fausse lecture et
         MOV     ES:[BP],BL        ; allumage du pixel.
         SUB     CX,8              ; CX = CX - 8
         JMP     SHORT R7          ; ... et on recommence.
```

Routine en Assembleur (suite)

```
                                        ;
R9:         MOV     AX,ØFFØ8H           ; Désactivation du masque
            OUT     DX,AX               ; (code FF registre 8).
            MOV     AX,ØØØ5H            ; Retour au mode normal d'écriture
            OUT     DX,AX               ; (code Ø registre 5).
            MOV     AX,ØØØ3H            ; Suppression du mode XOR
            OUT     DX,AX               ; (code ØØ registre 3).
                                        ;
            POP     DI                  ; Rétablissement index destination.
            POP     SI                  ; Rétablissement index source.
            POP     ES                  ; Rétablissement extra segment.
            POP     BP                  ; Rétablissement pointeur de base
            RET                         ; ... et retour à l'envoyeur.

_retic      ENDP
;------------------------------
CODE        ENDS
            END
```

Conçue d'emblée pour fonctionner en mode XOR, cette routine dessine d'abord le trait vertical suivi du trait horizontal.

Pour ce faire, elle calcule l'adresse de l'octet correspondant au haut du trait vertical (x_r , limite haute) et la passe à la sous-routine de dessin du trait vertical.

Ceci fait, elle calcule l'adresse de l'octet correspondant à l'extrémité gauche du trait horizontal (limite gauche, y_r) et la passe à son tour à la sous-routine de dessin du trait horizontal.

Notez qu'en ce qui concerne cette dernière, le dessin du trait horizontal s'effectue octet par octet, et non pixel par pixel, ce qui nécessite 3 phases de dessin selon qu'il s'agit de l'octet de début et de fin du trait (le plus souvent partiellement rempli...) et les autres octets qui eux, sont pleins.

Programme d'essai en Turbo C

```
int x,y;

void move_retic();

main()
{
  int mx,my;

  screen(16);
  xline(48Ø,Ø,48Ø,349,12);      /* Barre de menu */
  viewport(Ø,Ø,479,349);
  mx=24Ø,my=1ØØ;

  for(x=48Ø;x>239;x--){         /* Positionnement réticule */
    y=1ØØ;
    move_retic(3);
  }
  for(y=1ØØ; y<251; y++){       /* Dessin d´un triangle */
    x=24Ø-(y-1ØØ);
    xline(mx,my,x,y,1Ø);
    move_retic(3);
    mx=x;my=y;
  }
  for(x=9Ø; x<391; x++){
    y=25Ø;
    xline(mx,my,x,y,1Ø);
    move_retic(3);
    mx=x;my=y;
  }
  for(y=25Ø; y>99; y--){
    x=24Ø+(y-1ØØ);
    xline(mx,my,x,y,1Ø);
    move_retic(3);
    mx=x;my=y;
  }
  for(x=24Ø; x<481; x++){       /* Parkage du réticule */
    y=1ØØ;
    move_retic(3);
  }
}
```

Programme d'essai en Turbo C (suite)

```
void move_retic(couleur)
  int couleur;
  {
    int t;
    retic(x,y,couleur);
    for(t=0;t<1000;t++);
    retic(x,y,couleur);
  }
```

Destiné à simuler le dessin d'un triangle, ce petit bout d'essai dessine une barre de menu, délimite une fenêtre de travail, positionne le réticule sur la pointe du triangle, le dessine et retourne se cacher sous la barre de menu.

Notez que le déplacement du réticule n'altère aucunement le triangle en cours de dessin. C'est cela l'avantage du mode XOR.

18

COLORIER UN RECTANGLE EN XOR

A quoi cela peut-il bien servir de colorier une surface rectangulaire en mode XOR ?... Mais à mettre en relief un texte sur un fond de couleur, de façon à réaliser en page graphique des menus déroulants à la façon « Windows » (tout du moins, c'est à cela que nous l'avons employé...).

Routine en Assembleur

```
;          ******************************************************
;          Librairie ega.asm appelable à partir du Turbo C
;          ******************************************************

DATA       SEGMENT WORD 'DATA'

GAUCHE     DW      Ø              ; Limite gauche de la fenêtre.
HAUT       DW      Ø              ; Limite haute de la fenêtre.
DROITE     DW      639            ; Limite droite de la fenêtre.
BAS        DW      349            ; Limite basse de la fenêtre.
DELTAX     DW      ?              ; Deltax.
DELTAY     DW      ?              ; Deltay.
FLAG       DW      ?              ; Drapeau à tout faire.
XD         DW      ?              ; Sauvegarde de Xd.
XF         DW      ?              ; Sauvegarde de Xf.
YD         DW      ?              ; Sauvegarde de Yd.
YF         DW      ?              ; Sauvegarde de Yf.
```

Routine en Assembleur (suite)

```
DATA      ENDS
DGROUP    GROUP    DATA

CODE      SEGMENT BYTE 'CODE'
          ASSUME   CS:CODE
          ASSUME   DS:DGROUP
          PUBLIC   _screen
          PUBLIC   _viewport
          PUBLIC   _xline
          PUBLIC   _retic
          PUBLIC   _xfill
;------------------------------
_screen   PROC     FAR
           ;;
_screen   ENDP
;------------------------------
_viewport PROC     FAR
           ;;
_viewport ENDP
;------------------------------
_xline    PROC     FAR
           ;;
_xline    ENDP
;------------------------------
_retic    PROC     FAR
           ;;
_retic    ENDP
;------------------------------
_xfill    PROC     FAR
          PUSH     BP           ; Sauvegarde pointeur de base.
          MOV      BP,SP        ; Transfert pointeur de pile en BP.
          PUSH     ES           ; Sauvegarde extra segment.
          PUSH     SI           ; Sauvegarde index source.
          PUSH     DI           ; Sauvegarde index destination.
                                ;
          MOV      DX,0A000H    ; Chargement adresse de début
          MOV      ES,DX        ; dans extra segment (via DX).
          MOV      DX,3CEH      ; Port du contrôleur graphique.
          MOV      AX,0205H     ; Ecriture sur les 4 pans EGA.
          OUT      DX,AX        ; (mode 2 registre 5).
                                ;
          MOV      BX,(BP+0EH)  ; Tranfert couleur en BX.
          CMP      BX,128       ; XOR or not XOR ?
          JLE      F0           ; Non, alors c'est tout bon.
          MOV      AX,1803H     ; Passage en mode XOR
          OUT      DX,AX        ; (code 18 registre 3).
```

Routine en Assembleur (suite)

```
                                    ;
FØ:      MOV     SI,(BP+ØCH)        ; Transfert coordonnée Yf en SI.
         MOV     DI,(BP+ØAH)        ; Transfert coordonnée Xf en DI.
         MOV     AX,(BP+Ø8H)        ; Transfert coordonnée Yd en AX.
         MOV     CX,(BP+Ø6H)        ; Transfert coordonnée Xd en CX.
                                    ;
         CMP     DI,CX              ; Xf >= Xd ?
         JGE     F1                 ; Oui, alors c'est tout bon.
         XCHG    DI,CX              ; Non, alors on permute Xf avec Xd.
F1:      CMP     SI,AX              ; Yf >= Yd ?
         JGE     F2                 ; Oui, alors c'est tout bon.
         XCHG    SI,AX              ; Non, alors on permute Yf avec Yd.
                                    ;
F2:      SUB     DI,CX              ; Calcul de deltax.
         SUB     SI,AX              ; Calcul de deltay
         MOV     DELTAY,SI          ; et stockage en DELTAY.
                                    ;
         SHL     AX,1               ; Calcul de l'adresse de l'octet.
         SHL     AX,1               ; Multiplication par 8Ø de Yd.
         SHL     AX,1               ; 1re multiplication par 16
         SHL     AX,1               ; obtenue en décalant AX de 4 bits.
         MOV     BP,AX              ; Stockage du résultat en BP.
         SHL     AX,1               ; 2e multiplication par 64 obtenue
         SHL     AX,1               ; en décalant AX de 2 bits de plus.
         ADD     BP,AX              ; Addition du résultat à BP.
         MOV     AX,CX              ; Transfert de Xd en AX.
         SHR     AX,1               ; Division par 8 de Xd
         SHR     AX,1               ; réalisée en décalant AX
         SHR     AX,1               ; de 3 bits vers la droite.
         ADD     BP,AX              ; Adresse des 4 octets en BP.
         AND     CX,7               ; Position dans octets en CL.
                                    ;
         MOV     AX,ØFFØ8H          ; Chargement du masque plein
         SHR     AH,CL              ; et son décalage (via CL).
         XOR     CL,ØØØØØ111B       ; Calcul du complément à 7 du reste.
         SUB     CX,DI              ; Trait contenu dans octet ?
         JLE     F3                 ; Non, alors le masque est bon.
         SHR     AH,CL              ; Oui, alors mise en forme du masque
         SHL     AH,CL              ; grâce à un droite-gauche.
                                    ;
F3:      NEG     CX                 ; CX change de signe.
         OUT     DX,AX              ; Transfert du masque au contrôleur.
         XOR     DI,DI              ; Raz de DI.
```

Routine en Assembleur (suite et fin)

```
F4:     MOV     BH,ES:[BP+DI]; Fausse lecture pour vider la mémoire.
        MOV     ES:[BP+DI],BL; Allumage du pixel.
        ADD     DI,8Ø        ; DI = DI + 8Ø.
        DEC     SI           ; Deltay - 1 >= Ø ?
        JGE     F4           ; Oui, alors ligne suivante.
        MOV     AH,11111111B ; Non, alors on recharge le masque.
                             ;
F5:     CMP     CX,Ø         ; CX < Ø ?
        JL      F9           ; Oui, alors c'est fini.
F6:     CMP     CX,8         ; CX >= 8 ?
        JGE     F7           ; Oui, alors le masque est bon.
        SHR     AH,CL        ; Non, alors on le décale
        XOR     AH,11111111B ; et on le complémente.
                             ;
F7:     OUT     DX,AX        ; Transfert du masque au contrôleur.
        MOV     SI,DELTAY    ; On recharge deltay en SI.
        XOR     DI,DI        ; Raz de DI.
        INC     BP           ; Octet suivant.
F8:     MOV     BH,ES:[BP+DI]; Fausse lecture pour vider la mémoire.
        MOV     ES:[BP+DI],BL; Allumage du pixel.
        ADD     DI,8Ø        ; DI = DI + 8Ø.
        DEC     SI           ; Deltay - 1 >= Ø ?
        JGE     F8           ; Oui, alors ligne suivante.
        SUB     CX,8         ; CX = CX - 8.
        JMP     SHORT F5     ; ... et on recommence.
                             ;
F9:     MOV     AX,ØFFØ8H    ; Désactivation du masque
        OUT     DX,AX        ; (code FF registre 8).
        MOV     AX,ØØØ5H     ; Retour au mode normal d'écriture
        OUT     DX,AX        ; (code Ø registre 5).
        MOV     AX,ØØØ3H     ; Suppression du mode XOR
        OUT     DX,AX        ; (code ØØ registre 3).
                             ;
        POP     DI           ; Rétablissement index destination.
        POP     SI           ; Rétablissement index source.
        POP     ES           ; Rétablissement extra segment.
        POP     BP           ; Rétablissement pointeur de base
        RET                  ; ... et retour à l'envoyeur.

_xfill  ENDP
;-----------------------------
CODE    ENDS
        END
```

Sachant que le dessin des traits horizontaux est plus rapide que celui des traits verticaux (on dessine par octets complets et non par bits isolés...), il est préférable de considérer le rectangle à remplir comme une ligne horizontale épaisse, l'épaisseur étant précisément égale à la hauteur du rectangle.

Dans ces conditions, il suffit de remanier légèrement la portion de la routine retic dessinant les traits horizontaux en lui incorporant à chaque stade du dessin (début, milieu et fin du trait...) une boucle analogue à celle employée avec les traits verticaux.

Programme d'essai en Turbo C

```
int i;
main()
{
  screen(16);
  i=175;
  while(i>-1)
  {
    xfill(320-i,175-i,320+i,175+i,141);
    i=i-1;
    xfill(320-i,175-i,320+i,175+i,141);
    i=i-1;
  }
}
```

Petit mais costaud, ce programme dessine une suite de rectangles imbriqués les uns dans les autres, un rectangle sur deux étant affiché dans la couleur de fond de l'écran (noir pour la circonstance...).

Notez que cela s'obtient, non avec la couleur 0 mais, bien au contraire, en dessinant en XOR avec la même couleur.

19

AFFICHER UNE ICÔNE

La présence d'icônes dans une application graphique peut s'avérer d'un attrait important, ne serait-ce que parce qu'elles ouvrent la voie au dialogue symbolique homme-machine. C'est pourquoi vous trouverez ci-après une routine de dessin d'icônes qui accepte le chargement du motif de l'icône par le biais d'un tableau, restant ainsi ouverte à toute proposition.

Routine en Assembleur

```
;         ************************************************
;         Librairie ega.asm appelable à partir du Turbo C
;         ************************************************

DATA      SEGMENT WORD 'DATA'

GAUCHE    DW     Ø           ; Limite gauche de la fenêtre.
HAUT      DW     Ø           ; Limite haute de la fenêtre.
DROITE    DW     639         ; Limite droite de la fenêtre.
BAS       DW     349         ; Limite basse de la fenêtre.
DELTAX    DW     ?           ; Deltax.
DELTAY    DW     ?           ; Deltay.
FLAG      DW     ?           ; Drapeau à tout faire.
INDICE    DB     ?           ; Indice du masque d'icône.
MAX       DB     ?           ; Nombre de masques d'icône.
XD        DW     ?           ; Sauvegarde de Xd.
XF        DW     ?           ; Sauvegarde de Xf.
YD        DW     ?           ; Sauvegarde de Yd.
YF        DW     ?           ; Sauvegarde de Yf.
```

Routine en Assembleur (suite)

```
DATA      ENDS
DGROUP    GROUP   DATA

CODE      SEGMENT BYTE 'CODE'
          ASSUME  CS:CODE
          ASSUME  DS:DGROUP

          PUBLIC  _screen
          PUBLIC  _viewport
          PUBLIC  _xline
          PUBLIC  _retic
          PUBLIC  _xfill
          PUBLIC  _draw

;------------------------------
_screen   PROC    FAR
          ;;
_screen   ENDP
;------------------------------
_viewport PROC    FAR
          ;;
_viewport ENDP
;------------------------------
_xline    PROC    FAR
          ;;
_xline    ENDP
;------------------------------
_retic    PROC    FAR
          ;;
_retic    ENDP
;------------------------------
_xfill    PROC    FAR
          ;;
_xfill    ENDP
;------------------------------
_draw     PROC    FAR

          PUSH    BP          ; Sauvegarde pointeur de base.
          MOV     BP,SP       ; Transfert pointeur de pile en BP.
          PUSH    ES          ; Sauvegarde extra segment.
          PUSH    SI          ; Sauvegarde index source.
          PUSH    DI          ; Sauvegarde index destination.
                              ;
          MOV     DX,0A000H   ; Chargement adresse de début
          MOV     ES,DX       ; dans extra segment.
          MOV     DX,03CEH    ; Port du contrôleur graphique.
          MOV     AX,0205H    ; Ecriture sur les 4 plans EGA
          OUT     DX,AX       ; (code 2 registre 5).
```

Routine en Assembleur (suite)

```
                                ;
        MOV     DI,(BP+Ø6H)     ; Adresse du tableau en DI.
        MOV     AX,(BP+Ø8H)     ; Transfert du nombre
        MOV     MAX,AL          ; de masques en MAX.
        MOV     CX,(BP+ØAH)     ; Transfert coordonnée X en CX.
        MOV     AX,(BP+ØCH)     ; Transfert coordonnée Y en AX.
        MOV     BX,(BP+ØEH)     ; Chargement couleur en BX.
                                ;
        SHL     AX,1            ; Calcul de l'adresse de l'octet.
        SHL     AX,1            ; Multiplication par 8Ø de Y.
        SHL     AX,1            ; 1re multiplication par 16
        SHL     AX,1            ; obtenue en décalant AX de 4 bits.
        MOV     BP,AX           ; Stockage du résultat en BP.
        SHL     AX,1            ; 2e multiplication par 64 obtenue
        SHL     AX,1            ; en décalant AX de 2 bits de plus.
        ADD     BP,AX           ; Addition du résultat à BP.
        MOV     AX,CX           ; Transfert de X en AX.
        SHR     AX,1            ; Division par 8 de X
        SHR     AX,1            ; réalisée en décalant AX
        SHR     AX,1            ; de 3 bits vers la droite.
        ADD     BP,AX           ; Adresse des 4 octets en BP.
        AND     CL,ØØØØØ111B    ; Position dans octets en CL.
                                ;
        MOV     INDICE,Ø        ; Raz indice masque d'icône.
        XOR     SI,SI           ; Raz décalage ligne.
        MOV     AL,8            ; Registre 8 en AL.
                                ;
D1:     MOV     AH,[DI]         ; Chargement masque d'icône et
        ROR     AH,CL           ; mise en forme par CL.
        MOV     CH,11111111B    ; Chargement masque plein et
        SHR     CH,CL           ; positionnement pour 1er octet.
        AND     AH,CH           ; Masquage avec masque d'icône.
        OUT     DX,AX           ; Transfert du masque au contrôleur.
        MOV     BH,ES:[BP+SI]; Fausse lecture pour vider la mémoire.
        MOV     ES:[BP+SI],BL; Allumage 1er octet.
        CMP     CL,Ø            ; Y a-t-il un 2nd octet ?
        JE      D2              ; Non.
                                ;
        MOV     AH,[DI]         ; Chargement masque d'icône et
        ROR     AH,CL           ; mise en forme par CL (bis).
        XOR     CH,11111111B    ; Positionnement pour 2nd octet.
        AND     AH,CH           ; Masquage avec masque d'icône.
        OUT     DX,AX           ; Transfert du masque au contrôleur.
        MOV     BH,ES:[BP+1+SI]; Fausse lecture pour vider la mémoire.
        MOV     ES:[BP+1+SI],BL; Allumage 2nd octet.
```

Routine en Assembleur (suite)

```
                                  ;
D2:        ADD    SI,80           ; Ligne suivante.
           ADD    DI,2            ; Incrémentation adresse.
           INC    INDICE          ; Masque d'icône suivant.
           MOV    BH,INDICE       ; Prélude à la comparaison.
           CMP    BH,MAX          ; A-t-on dessiné les 12 masques ?
           JL     D1              ; Non, alors on recommence.
                                  ;
           MOV    AX,0FF08H       ; Désactivation du masque
           OUT    DX,AX           ; (code FF registre 8).
           MOV    AX,0005H        ; Retour au mode normal d'écriture
           OUT    DX,AX           ; (code 00 registre 5).
                                  ;
           POP    DI              ; Rétablissement index destinataire.
           POP    SI              ; Rétablissement index source.
           POP    ES              ; Rétablissement extra segment.
           POP    BP              ; Rétablissement pointeur de base
           RET                    ; ... et retour à l'envoyeur.

_draw      ENDP
;-----------------------------
CODE       ENDS
           END
```

Après les habituels sauvegardes de registres, chargements de paramètres et calculs d'adresse, on procède au chargement dans DI de l'adresse du masque d'indice 0, le registre DI servant alors de pointeur d'adresse.

On charge alors le premier masque d'icône en AH et on le positionne (sauf si CL vaut zéro...). Pourquoi faut-il positionner le masque d'icône ?... Parce que, selon la position de l'icône sur l'écran, celle-ci a 7 chances sur 8 d'être à cheval sur 2 octets consécutifs. Il convient donc de déterminer ce qui doit être dessiné dans chacun des 2 octets et c'est pourquoi le masque tourne...

Le masque d'icône ayant tourné, on met alors en forme le masque du 1er octet selon une technique éprouvée (chargement masque plein, décalage selon CL...), auquel on superpose le masque d'icône. On est alors prêt à transférer dans la mémoire EGA le contenu du 1er octet.

Le mode opératoire pour le 2nd octet (à condition qu'il y en ait un...) est exactement le même, à savoir qu'on recharge le masque d'icône en AH, qu'on le fait tourner comme précédemment, qu'on met en forme le masque du 2nd octet (avec un XOR cette fois-ci...) et qu'on expédie le tout à l'adresse BP + 1.

Et quand on a fini avec le premier masque d'icône, on passe aux suivants. Notez l'emploi des variables INDICE et MAX pour repérer l'indice du masque d'icône en cours d'affichage et le nombre maximal de masques à traiter.

Notez que chaque masque est long de 8 bits, longueur qui peut être portée à 16 bits (ou même plus...) si le motif de l'icône le nécessite. Il suffit pour cela d'effectuer la rotation des masques sur 3 octets et non 2 comme c'est le cas présentement.

Programme d'essai en Turbo C

```
main()
{
int main[9];

  main[Ø] = ØxCØ;              /* Masques de la main */
  main[1] = Øx6Ø;
  main[2] = Øx33;
  main[3] = Øx1A;
  main[4] = Øx7E;
  main[5] = ØxFF;
  main[6] = ØxFE;
  main[7] = Øx7D;
  main[8] = Øx3B;
  main[9] = Øx16;

screen(16);
draw(&main,1Ø,32Ø,175,15);     /* Dessin de la main */
}
```

Que fait ce programme ?... Il affiche une main (à ne pas confondre avec la déclaration du même nom...) en plein milieu de l'écran, les 10 masques de la main étant déclarés dans un tableau.

Notez la transmission de l'adresse de l'indice 0 du tableau à la routine (pointeur d'adresse &), suivie du nombre d'indices du tableau (et donc de masques...) à transmettre.

20

PROTÉGER UN FICHIER CONTRE L'EFFACEMENT

Après avoir effacé par mégarde un fichier, n'avez-vous jamais regretté de ne pas disposer des moyens d'interdire cet effacement ?... Nous, si !... C'est pourquoi nous vous livrons l'utilitaire que voici qui, baptisé lock pour la circonstance, trouvera sa place parmi les commandes usuelles du DOS.

Routine en Assembleur

```
;          ******************************************
;          Routine lock.asm écrite pour fichier .COM
;          ******************************************

CODE       SEGMENT BYTE 'CODE'
           ASSUME  CS:CODE
           ASSUME  DS:CODE

           ORG     100H

lock:      JMP     debut

fichier:   DB      16 DUP (00)

erreur1:   DB      07h,'Nom de fichier absent...',0dh,0ah,0ah
           DB      '╔═══════════╤══════════════════════╗',0dh,0ah
           DB      '║   LOCK    │  Protège un fichier  ║',0dh,0ah
           DB      '╚═══════════╧══════════════════════╝',0dh,0ah,0ah
```

Routine en Assembleur (suite)

```
            DB      'protège contre l',27h,'effacement tout fichier:',Ødh,Øah,Øah
            DB      '           - visible',Ødh,Øah
            DB      '           - invisible',Ødh,Øah
            DB      '           - système',Ødh,Øah
            DB      '           - archive',Ødh,Øah,Øah,Øah
            DB      'Tapez votre commande sous la forme:',Ødh,Øah,Øah
            DB      '             LOCK [u:][chemin]nomfichier[.ext]',Ødh,Øah,Øah
            DB      'Exemples: - LOCK LISTE.TXT',Ødh,Øah
            DB      '           - LOCK A:CALCUL.BAS',Ødh,Øah
            DB      '           - LOCK C:\SOCIETE\BILAN.ASC',Ødh,Øah,Øah,24h

erreur2:    DB      'Fichier non trouvé...',Ødh,Øah,24h
erreur3:    DB      'Fichier déjà protégé...',Ødh,Øah,24h
erreur4:    DB      'Fichier protégé...',Ødh,Øah,24h

debut:      MOV     SI,8ØH              ; Lecture 1er octet du PSP
            LODSB                       ; (décalage 8ØH).
            OR      AL,AL               ; Est-ce un zéro ?
            JNZ     L1                  ; Non, alors c'est tout bon.
            MOV     DX,OFFSET erreur1;  Oui, alors message
            JMP     SHORT exit          ; "Nom de fichier absent".
                                        ;
L1:         LODSB                       ; Lecture caractère suivant.
            CMP     AL,2ØH              ; Est-ce un espace ?
            JE      L1                  ; Oui, alors on recommence.
            DEC     SI                  ; Non, alors on positionne les
            MOV     DI,OFFSET fichier;  index SI et DI pour le transfert.
            MOV     DX,DI               ; Début nom de fichier en DX.
                                        ;
L2:         MOVSB                       ; Transfert du nom du fichier
            CMP     BYTE PTR [SI],Ødh;  dans la zone réservée.
            JNE     L2                  ; Retour-chariot = fin.
                                        ;
            MOV     AH,4EH              ; Attente d'un nom de fichier.
            INT     21H                 ; Ce nom est-il correct ?
            JNC     L3                  ; Oui, alors c'est tout bon.
            MOV     DX,OFFSET erreur2;  Non, alors message
            JMP     SHORT exit          ; "Fichier non trouvé".
                                        ;
L3:         MOV     CL,DS:[ØØ95H]       ; Octet d'attribut en CL.
            TEST    CL,ØØØØØØØ1B        ; Fichier déjà protégé ?
            JE      L4                  ; Non, alors on protège.
            MOV     DX,OFFSET erreur3;  Oui, alors message
            JMP     SHORT exit          ; "Fichier déjà protégé".
```

Routine en Assembleur (suite)

```
                                      ;
L4:       OR      CL,00000001B        ; Forçage à 1 du bit n° 0
          MOV     AX,4301H            ; de l'octet d'attribut.
          INT     21H                 ; (fonction 43-1, interruption 21).
          MOV     DX,OFFSET erreur4;  Message "Fichier protégé".
                                      ;
exit:     MOV     AH,09H              ; Affichage du message
          INT     21H                 ; (fonction 9, interruption 21)
          INT     20H                 ; ... et retour au DOS.

CODE      ENDS
          END     lock
```

La protection contre l'effacement s'obtient en forçant à 1 le bit n° 0 de l'octet d'attribut du fichier, ce que réalise fort bien la sous-fonction 1 de la fonction 43 de l'interruption 21.

La routine réalisant cela en 3 lignes, le reste de celle-ci sert à lire le nom du fichier, à le rechercher et à gérer les messages d'erreur éventuels.

Après déclaration du registre-tampon d'entrée (de 16 octets de long...) et des 4 messages d'erreur, la routine débute par la lecture du 1er octet du PSP. Si ce dernier est un zéro, ce qui correspond à l'oubli du nom du fichier à la suite de la commande lock, le message « Nom de fichier absent » est affiché, suivi d'un écran d'aide.

Ceci fait, la routine recherche les espaces superflus (> 1) précédant le nom de fichier et les élimine avant le transfert de ce dernier dans le registre-tampon qui lui est attribué. Notez au passage que le transfert s'arrête dès la présence d'un retour-chariot dans le registre AL.

La recherche du fichier s'effectue alors par l'intermédiaire de la fonction 4E de l'interruption 21, le résultat de la recherche figurant dans la retenue (CF = 0 si fichier trouvé, CF = 1 dans le cas contraire). Notez l'envoi du message «Fichier non trouvé » lorsque la retenue vaut 1.

L'octet d'attribut du fichier ayant été chargé à l'adresse 95 du PSP, il suffit de le transférer dans le registre CL de façon à déterminer l'état du bit n° 0 (0 = fichier non protégé, 1 = fichier protégé). Si le fichier est déjà protégé, on le signale par le message « Fichier déjà protégé », sinon on force à 1 ce bit de protection et on signale que le fichier est désormais protégé.

Point intéressant : *La sortie de routine s'accompagnant obligatoirement d'un message d'erreur, son affichage n'est à prévoir qu'une seule fois, ce qui allège l'écriture de la routine.*

Création d'un fichier .COM

Destinée à devenir autonome sous la forme d'un fichier .COM, cette routine débute par l'inévitable ORG 100H.

Notez l'absence de déclaration de type public et de procédure FAR, avec ce que cela implique de sauvegarde du pointeur de base et de transfert du pointeur de pile en BP. Ici, tout cela est inutile !... Il suffit de déclarer le segment de code, un point c'est (presque...) tout !

Notez quand même la zone des messages d'erreur située en début de routine qu'un JMP court-circuite allègrement. Notez également l'emploi des étiquettes lock en début et en fin de routine.

Ceci, dit, il reste à obtenir un fichier .COM à partir du code source de cette routine. Pour ce faire, il vous faut lancer comme suit les 3 utilitaires que voici :

```
>MASM LOCK;
>LINK LOCK;
>EXE2BIN LOCK LOCK.COM
```

Le résultat est un fichier LOCK.COM de 630 octets de long qui peut désormais être lancé à partir du DOS.

21

DÉVERROUILLER UN FICHIER PROTÉGÉ

Complémentaire de la précédente, la routine unlock que voici supprime la protection d'effacement des fichiers protégés. De surcroît, elle fonctionne sur tout type de fichier, qu'il s'agisse de fichiers visibles, cachés, systèmes ou archives.

Routine en Assembleur

```
;          ********************************************
;          Routine unlock.asm écrite pour fichier .COM
;          ********************************************

CODE       SEGMENT BYTE 'CODE'
           ASSUME  CS:CODE
           ASSUME  DS:CODE

           ORG     100H

unlock:    JMP     debut

fichier:   DB      16 DUP (00)

erreur1:   DB      07h,'Nom de fichier absent...',0dh,0ah,0ah
           DB      '╔══════════════════════════════════════════╗',0dh,0ah
           DB      '║ UNLOCK  │ Déverrouille un fichier protégé ║',0dh,0ah
           DB      '╚══════════════════════════════════════════╝',0dh,0ah,0ah
           DB      'Ce programme supprime la protection d',27h
           DB      'effacement des fichiers:',0dh,0ah,0ah
```

Routine en Assembleur (suite)

```
          DB      '            - visibles',Ødh,Øah
          DB      '            - invisibles',Ødh,Øah
          DB      '            - systèmes',Ødh,Øah
          DB      '            - archives',Ødh,Øah,Øah,Øah
          DB      'Tapez votre commande sous la forme:',Ødh,Øah,Øah
          DB      '               UNLOCK [u:][chemin]nomfichier[.ext]',Ødh,Øah,Øah
          DB      'Exemples:   - UNLOCK LISTE.TXT',Ødh,Øah
          DB      '            - UNLOCK A:CALCUL.BAS',Ødh,Øah
          DB      '            - UNLOCK C:\SOCIETE\BILAN.ASC',Ødh,Øah,Øah,24h

erreur2:  DB      'Fichier non trouvé...',Ødh,Øah,24h
erreur3:  DB      'Fichier déjà déverrouillé...',Ødh,Øah,24h
erreur4:  DB      'Fichier déverrouillé...',Ødh,Øah,24h

debut:    MOV     SI,8ØH              ; Lecture 1er octet du PSP
          LODSB                       ; (décalage 8ØH).
          OR      AL,AL               ; Est-ce un zéro ?
          JNZ     L1                  ; Non, alors c'est tout bon.
          MOV     DX,OFFSET erreur1; Oui, alors message
          JMP     SHORT exit          ; "Nom de fichier absent".
                                      ;
L1:       LODSB                       ; Lecture caractère suivant.
          CMP     AL,2ØH              ; Est-ce un espace ?
          JE      L1                  ; Oui, alors on recommence.
          DEC     SI                  ; Non, alors on positionne les
          MOV     DI,OFFSET fichier; index SI et DI pour le transfert.
          MOV     DX,DI               ; Début nom de fichier en DX.
                                      ;
L2:       MOVSB                       ; Transfert du nom du fichier
          CMP     BYTE PTR [SI],Ødh; dans la zone réservée.
          JNE     L2                  ; Retour-chariot = fin.
                                      ;
          MOV     AH,4EH              ; Attente d'un nom de fichier.
          INT     21H                 ; Ce nom est-il correct ?
          JNC     L3                  ; Oui, alors c'est tout bon.
          MOV     DX,OFFSET erreur2; Non, alors message
          JMP     SHORT exit          ; "Fichier non trouvé".
                                      ;
L3:       MOV     CL,DS:[ØØ95H]       ; Octet d'attribut en CL.
          TEST    CL,ØØØØØØØ1B        ; Fichier déjà déverrouillé ?
          JNE     L4                  ; Non, alors on déverrouille.
          MOV     DX,OFFSET erreur3; Oui, alors message
          JMP     SHORT exit          ; "Fichier déjà déverrouillé".
```

Routine en Assembleur (suite)

```
                                    ;
L4:         AND     CL,11111110B    ; Forçage à Ø du bit n° Ø
            MOV     AX,43Ø1H        ; de l'octet d'attribut.
            INT     21H             ; (fonction 43-1, interruption 21).
            MOV     DX,OFFSET erreur4; Message "Fichier déverrouillé".
                                    ;
exit:       MOV     AH,Ø9H          ; Affichage du message
            INT     21H             ; (fonction 9, interruption 21)
            INT     2ØH             ; ... et retour au DOS.

CODE        ENDS
            END     unlock
```

La protection contre l'effacement s'annule en forçant à 0 le bit n° 0 de l'octet d'attribut du fichier, ce que réalise fort bien la sous-fonction 1 de la fonction 43 de l'interruption 21.

La routine réalisant cela en 3 lignes, le reste de celle-ci sert à lire le nom du fichier, à le rechercher et à gérer les messages d'erreur éventuels.

Après déclaration du registre-tampon d'entrée (de 16 octets de long...) et des 4 messages d'erreur, la routine débute par la lecture du 1er octet du PSP. Si ce dernier est un zéro, ce qui correspond à l'oubli du nom du fichier à la suite de la commande unlock, le message « Nom de fichier absent » est affiché, suivi d'un écran d'aide.

Ceci fait, la routine recherche les espaces superflus (> 1) précédant le nom de fichier et les élimine avant le transfert de ce dernier dans le registre-tampon qui lui est attribué. Notez au passage que le transfert s'arrête dès la présence d'un retour-chariot dans le registre AL.

La recherche du fichier s'effectue alors par l'intermédiaire de la fonction 4F de l'interruption 21, le résultat de la recherche figurant dans la retenue (CF = 0 si fichier trouvé, CF = 1 dans le cas contraire). Notez l'envoi du message « Fichier non trouvé » lorsque la retenue vaut 1.

L'octet d'attribut du fichier ayant été chargé à l'adresse 95 du PSP, il suffit de le transférer dans le registre CL de façon à déterminer l'état du bit n° 0 (0 = fichier non protégé, 1 = fichier protégé). Si le fichier est déjà déverrouillé, on le signale par le message «Fichier déjà déverrouillé », sinon on force à 0 ce bit de protection et on signale que le fichier est désormais déverrouillé.

Point intéressant : *La sortie de routine s'accompagnant obligatoirement d'un message d'erreur, son affichage n'est à prévoir qu'une seule fois, ce qui allège l'écriture de la routine.*

Création d'un fichier .COM

Destinée à devenir autonome sous la forme d'un fichier .COM, cette routine débute par l'inévitable ORG 100H.

Notez l'absence de déclaration de type public et de procédure FAR, avec ce que cela implique de sauvegarde du pointeur de base et de transfert du pointeur de pile en BP. Ici, tout cela est inutile !... Il suffit de déclarer le segment de code, un point c'est (presque...) tout !

Notez quand même la zone des messages d'erreur située en début de routine qu'un JMP court-circuite allègrement. Notez également l'emploi des étiquettes unlock en début et en fin de routine.

Ceci dit, il reste à obtenir un fichier .COM à partir du code source de cette routine. Pour ce faire, il vous faut lancer comme suit les 3 utilitaires que voici :

```
>MASM UNLOCK;

>LINK UNLOCK;

>EXE2BIN UNLOCK UNLOCK.COM
```

Le résultat est un fichier UNLOCK.COM de 724 octets de long qui peut désormais être lancé à partir du DOS.

22

CACHER UN FICHIER

N'avez-vous jamais eu envie de rendre invisibles certains fichiers importants de façon à les soustraire aux yeux des curieux ?... Nous, si !... C'est pourquoi nous vous livrons l'utilitaire que voici qui, baptisé hide pour la circonstance, trouvera aisément sa place parmi les autres commandes du DOS.

Routine en Assembleur

```
;           ****************************************
;           Routine hide.asm écrite pour fichier .COM
;           ****************************************

CODE        SEGMENT BYTE 'CODE'
            ASSUME  CS:CODE
            ASSUME  DS:CODE

            ORG     100H

hide:       JMP     debut

fichier:    DB      16 DUP (00)

erreur1:    DB      07h,'Nom de fichier absent...',0dh,0ah,0ah
            DB      '╔═══════════╦═══════════════════╗',0dh,0ah
            DB      '║   HIDE    ║ Cache un fichier  ║',0dh,0ah
            DB      '╚═══════════╩═══════════════════╝',0dh,0ah,0ah
            DB      'Ce programme rend invisible tout fichier:',0dh,0ah,0ah
            DB      '           - normal',0dh,0ah
            DB      '           - protégé',0dh,0ah
            DB      '           - système',0dh,0ah
            DB      '           - archive',0dh,0ah,0ah,0ah
```

Routine en Assembleur (suite)

```
          DB     'Tapez votre commande sous la forme:',0dh,0ah,0ah
          DB     '            HIDE [u:][chemin]nomfichier[.ext]',0dh,0ah,0ah
          DB     'Exemples: - HIDE LISTE.TXT',0dh,0ah
          DB     '          - HIDE A:CALCUL.BAS',0dh,0ah
          DB     '          - HIDE C:\SOCIETE\BILAN.ASC',0dh,0ah,0ah,24h

erreur2:  DB     'Fichier non trouvé...',0dh,0ah,24h
erreur3:  DB     'Fichier déjà caché...',0dh,0ah,24h
erreur4:  DB     'Fichier caché...',0dh,0ah,24h

debut:    MOV    SI,80H             ; Lecture 1er octet du PSP
          LODSB                     ; (décalage 80H).
          OR     AL,AL              ; Est-ce un zéro ?
          JNZ    L1                 ; Non, alors c'est tout bon.
          MOV    DX,OFFSET erreur1; Oui, alors message
          JMP    SHORT exit         ; "Nom de fichier absent".
                                    ;
L1:       LODSB                     ; Lecture caractère suivant.
          CMP    AL,20H             ; Est-ce un espace ?
          JE     L1                 ; Oui, alors on recommence.
          DEC    SI                 ; Non, alors on positionne les
          MOV    DI,OFFSET fichier; index SI et DI pour le transfert.
          MOV    DX,DI              ; Début nom de fichier en DX.
                                    ;
L2:       MOVSB                     ; Transfert du nom du fichier
          CMP    BYTE PTR [SI],0dh; dans la zone réservée.
          JNE    L2                 ; Retour-chariot = fin.
                                    ;
          MOV    AH,4EH             ; Attente d'un nom de fichier.
          INT    21H                ; Ce nom est-il correct ?
          JNC    L3                 ; Oui, alors c'est tout bon.
          MOV    DX,OFFSET erreur2; Non, alors message
          JMP    SHORT exit         ; "Fichier non trouvé".
                                    ;
L3:       MOV    CL,DS:[0095H]      ; Octet d'attribut en CL.
          TEST   CL,00000010B       ; Fichier déjà caché ?
          JE     L4                 ; Non, alors on le cache.
          MOV    DX,OFFSET erreur3; Oui, alors message
          JMP    SHORT exit         ; "Fichier déjà caché".
                                    ;
L4:       OR     CL,00000010B       ; Forçage à 1 du bit n°1
          MOV    AX,4301H           ; de l'octet d'attribut.
          INT    21H                ; (fonction 43-1, interruption 21).
          MOV    DX,OFFSET erreur4; Message "Fichier protégé".
```

Routine en Assembleur (suite)

```
                                        ;
exit:       MOV     AH,09H              ; Affichage du message
            INT     21H                 ; (fonction 9, interruption 21)
            INT     20H                 ; ... et retour au DOS.

CODE        ENDS
            END     hide
```

L'invisibilité d'un fichier s'obtient en forçant à 1 le bit n° 1 de l'octet d'attribut du fichier, ce que réalise fort bien la sous-fonction 1 de la fonction 43 de l'interruption 21.

La routine réalisant cela en 3 lignes, le reste de celle-ci sert à lire le nom du fichier, à le rechercher et à gérer les messages d'erreur éventuels.

Après déclaration du registre-tampon d'entrée (de 16 octets de long...) et des 4 messages d'erreur, la routine débute par la lecture du 1er octet du PSP. Si ce dernier est un zéro, ce qui correspond à l'oubli du nom du fichier à la suite de la commande hide, le message « Nom de fichier absent » est affiché, suivi d'un écran d'aide.

Ceci fait, la routine recherche les espaces superflus (> 1) précédant le nom de fichier et les élimine avant le transfert de ce dernier dans le registre-tampon qui lui est attribué. Notez au passage que le transfert s'arrête dès la présence d'un retour-chariot dans le registre AL.

La recherche du fichier s'effectue alors par l'intermédiaire de la fonction 4E de l'interruption 21, le résultat de la recherche figurant dans la retenue (CF = 0 si fichier trouvé, CF = 1 dans le cas contraire). Notez l'envoi du message « Fichier non trouvé » lorsque la retenue vaut 1.

L'octet d'attribut du fichier ayant été chargé à l'adresse 95 du PSP, il suffit de le transférer dans le registre CL de façon à déterminer l'état du bit n° 1 (0 = fichier non caché, 1 = fichier caché). Si le fichier est déjà caché, on le signale par le message « Fichier déjà caché », sinon on force à 1 ce bit d'invisibilité et on signale que le fichier est désormais caché.

Point intéressant : *La sortie de routine s'accompagnant obligatoirement d'un message d'erreur, son affichage n'est à prévoir qu'une seule fois, ce qui allège l'écriture de la routine.*

Création d'un fichier .COM

Destinée à devenir autonome sous la forme d'un fichier .COM, cette routine débute par l'inévitable ORG 100H.

Notez l'absence de déclaration de type public et de procédure FAR, avec ce que cela implique de sauvegarde du pointeur de base et de transfert du pointeur de pile en BP. Ici, tout cela est inutile !... Il suffit de déclarer le segment de code, un point c'est (presque...) tout !

Notez quand même la zone des messages d'erreur située en début de routine qu'un JMP court-circuite allègrement. Notez également l'emploi des étiquettes hide en début et en fin de routine.

Ceci dit, il reste à obtenir un fichier .COM à partir du code source de cette routine. Pour ce faire, il vous faut lancer comme suit les 3 utilitaires que voici :

```
>MASM HIDE;

>LINK HIDE;

>EXE2BIN HIDE HIDE.COM
```

Le résultat est un fichier HIDE.COM de 604 octets de long qui peut désormais être lancé à partir du DOS.

23

RENDRE VISIBLE UN FICHIER CACHÉ

Complémentaire de la précédente, la routine show que voici rend de nouveau visible un fichier caché. Notez qu'elle fonctionne sur tout type de fichier, qu'il s'agisse de fichiers normaux, protégés, systèmes ou archives.

Routine en Assembleur

```
;           ******************************************
;           Routine show.asm écrite pour fichier .COM
;           ******************************************

CODE        SEGMENT BYTE 'CODE'
            ASSUME  CS:CODE
            ASSUME  DS:CODE

            ORG     100H

show:       JMP     debut

fichier:    DB      16 DUP (00)

erreur1:    DB      07h,'Nom de fichier absent...',0dh,0ah,0ah
            DB      '╔══════════╤═══════════════════════════════╗',0dh,0ah
            DB      '║  SHOW    │ Rend visible un fichier caché ║',0dh,0ah
            DB      '╚══════════╧═══════════════════════════════╝',0dh,0ah,0ah
            DB      'Ce programme rend visible tout fichier:',0dh,0ah,0ah
            DB      '           - normal',0dh,0ah
            DB      '           - protégé',0dh,0ah
            DB      '           - système',0dh,0ah
            DB      '           - archive',0dh,0ah,0ah,0ah
```

Routine en Assembleur (suite)

```
            DB      'Tapez votre commande sous la forme:',Ødh,Øah,Øah
            DB      '               SHOW [u:][chemin]nomfichier[.ext]',Ødh,Øah,Øah
            DB      'Exemples: - SHOW LISTE.TXT',Ødh,Øah
            DB      '          - SHOW A:CALCUL.BAS',Ødh,Øah
            DB      '          - SHOW C:\SOCIETE\BILAN.ASC',Ødh,Øah,Øah,24h

erreur2:    DB      'Fichier non trouvé...',Ødh,Øah,24h
erreur3:    DB      'Fichier déjà visible...',Ødh,Øah,24h
erreur4:    DB      'Fichier visible...',Ødh,Øah,24h

debut:      MOV     SI,8ØH              ; Lecture 1er octet du PSP
            LODSB                       ; (décalage 8ØH).
            OR      AL,AL               ; Est-ce un zéro ?
            JNZ     L1                  ; Non, alors c'est tout bon.
            MOV     DX,OFFSET erreur1; Oui, alors message
            JMP     SHORT exit          ; "Nom de fichier absent".
                                        ;
L1:         LODSB                       ; Lecture caractère suivant.
            CMP     AL,2ØH              ; Est-ce un espace ?
            JE      L1                  ; Oui, alors on recommence.
            DEC     SI                  ; Non, alors on positionne les
            MOV     DI,OFFSET fichier; index SI et DI pour le transfert.
            MOV     DX,DI               ; Début nom de fichier en DX.
                                        ;
L2:         MOVSB                       ; Transfert du nom du fichier
            CMP     BYTE PTR [SI],Ødh; dans la zone réservée.
            JNE     L2                  ; Retour-chariot = fin.
                                        ;
            MOV     AH,4EH              ; Attente d'un nom de fichier.
            INT     21H                 ; Ce nom est-il correct ?
            JNC     L3                  ; Oui, alors c'est tout bon.
            MOV     DX,OFFSET erreur2; Non, alors message
            JMP     SHORT exit          ; "Fichier non trouvé".
                                        ;
L3:         MOV     CL,DS:[ØØ95H]       ; Octet d'attribut en CL.
            TEST    CL,ØØØØØ11ØB        ; Fichier déjà visible ?
            JNE     L4                  ; Non, alors on le rend visible.
            MOV     DX,OFFSET erreur3; Oui, alors message
            JMP     SHORT exit          ; "Fichier déjà visible".
                                        ;
L4:         AND     CL,11111ØØ1B        ; Forçage à Ø des bits n°1 et 2
            MOV     AX,43Ø1H            ; de l'octet d'attribut.
            INT     21H                 ; (fonction 43-1, interruption 21).
            MOV     DX,OFFSET erreur4; Message "Fichier visible".
```

Routine en Assembleur (suite)

```
                                        ;
exit:     MOV     AH,Ø9H                ; Affichage du message
          INT     21H                   ; (fonction 9, interruption 21)
          INT     2ØH                   ; ... et retour au DOS.

CODE      ENDS
          END     show
```

La visibilité d'un fichier s'obtient en forçant à 00 les bits n° 1 et 2 de l'octet d'attribut du fichier, ce que réalise fort bien la sous-fonction 1 de la fonction 43 de l'interruption 21.

La routine réalisant cela en 3 lignes, le reste de celle-ci sert à lire le nom du fichier, à le rechercher et à gérer les messages d'erreur éventuels.

Après déclaration du registre-tampon d'entrée (de 16 octets de long...) et des 4 messages d'erreur, la routine débute par la lecture du 1er octet du PSP. Si ce dernier est un zéro, ce qui correspond à l'oubli du nom du fichier à la suite de la commande show, le message « Nom de fichier absent » est affiché, suivi d'un écran d'aide.

Ceci fait, la routine recherche les espaces superflus (> 1) précédant le nom de fichier et les élimine avant le transfert de ce dernier dans le registre-tampon qui lui est attribué. Notez au passage que le transfert s'arrête dès la présence d'un retour-chariot dans le registre AL.

La recherche du fichier s'effectue alors par l'intermédiaire de la fonction 4E de l'interruption 21, le résultat de la recherche figurant dans la retenue (CF = 0 si fichier trouvé, CF = 1 dans le cas contraire). Notez l'envoi du message « Fichier non trouvé » lorsque la retenue vaut 1.

L'octet d'attribut du fichier ayant été chargé à l'adresse 95 du PSP, il suffit de le transférer dans le registre CL de façon à déterminer l'état des bits n° 1 et 2 (00 = fichier visible). Si le fichier est déjà visible, on le signale par le message « Fichier

déjà visible », sinon on force à 00 les bits n° 1 et 2 et on signale que le fichier est désormais visible.

Point intéressant : ***La sortie de routine s'accompagnant obligatoirement d'un message d'erreur, son affichage n'est à prévoir qu'une seule fois, ce qui allège l'écriture de la routine.***

Création d'un fichier .COM

Destinée à devenir autonome sous la forme d'un fichier .COM, cette routine débute par l'inévitable ORG 100H.

Notez l'absence de déclaration de type public et de procédure FAR, avec ce que cela implique de sauvegarde du pointeur de base et de transfert du pointeur de pile en BP. Ici, tout cela est inutile !... Il suffit de déclarer le segment de code, un point c'est (presque...) tout !

Notez quand même la zone des messages d'erreur située en début de routine qu'un JMP court-circuite allègrement. Notez également l'emploi des étiquettes show en début et en fin de routine.

Ceci dit, il reste à obtenir un fichier .COM à partir du code source de cette routine. Pour ce faire, il vous faut lancer comme suit les 3 utilitaires que voici :

```
>MASM SHOW;

>LINK SHOW;

>EXE2BIN SHOW SHOW.COM
```

Le résultat est un fichier SHOW.COM de 648 octets de long qui peut désormais être lancé à partir du DOS.

24

SAUVEGARDER UNE IMAGE EGA

Sauvegarder une image EGA en un fichier binaire est une opération qui ne savent pas (encore...) réaliser les langages actuels de développement. C'est pourquoi nous vous livrons cette routine egasave qui, fruit de nos efforts, vous permettra désormais de sauvegarder vos images EGA en toute facilité.

Routine en Assembleur

```
;           *************************************************
;           Routine egasave.asm appelable à partir du Turbo C
;           *************************************************

CODE        SEGMENT BYTE 'CODE'
            ASSUME  CS:CODE
            PUBLIC  _egasave
_egasave    PROC    FAR

            PUSH    BP                  ; Sauvegarde pointeur de base.
            MOV     BP,SP               ; Transfert pointeur de pile en BP.
            PUSH    DS                  ; Sauvegarde segment de données.
                                        ;
            MOV     DX,(BP+Ø6H)         ; Adresse nom de fichier en DX.
            XOR     CX,CX               ; Attribut nul.
            MOV     AH,3CH              ; Création du fichier d'image
            INT     21H                 ; (fonction 3C, interruption 21).
```

Routine en Assembleur (suite)

```
                                          ;
            MOV     BX,AX                 ; Transfert témoin de fichier en BX.
            MOV     CX,28000              ; Nombre d'octets par plan.
                                          ;
            MOV     DX,03CEH              ; Port du contrôleur graphique.
            MOV     AX,0004H              ; Sélection du plan n°0
E1:         OUT     DX,AX                 ; (code 0, registre 4).
            PUSH    DX                    ; Sauvegarde adresse contrôleur.
            PUSH    AX                    ; Sauvegarde n° de plan.
                                          ;
            MOV     AX,0A000H             ; Chargement adresse de début
            MOV     DS,AX                 ; dans segment de données.
            XOR     DX,DX                 ; Raz décalage.
            MOV     AH,40H                ; Ecriture dans fichier d'image
            INT     21H                   ; (fonction 40, interruption 21).
                                          ;
            POP     AX                    ; Rétablissement n° de plan.
            POP     DX                    ; Rétablissement adresse contrôleur.
            INC     AH                    ; Incrémentation n° de plan.
            CMP     AH,4                  ; A-t-on sauvegardé les 4 plans ?
            JL      E1                    ; Non, alors on continue.
                                          ;
            MOV     AH,3EH                ; Clotûre du fichier d'image
            INT     21H                   ; (fonction 3E, interruption 21).
                                          ;
            MOV     AX,0004H              ; Rétablissement du plan n°0
            OUT     DX,AX                 ; (code 0, registre 4).
                                          ;
            POP     DS                    ; Rétablissement segment de données.
            POP     BP                    ; Rétablissement pointeur de base
            RET                           ; ... et retour à l'envoyeur.

_egasave    ENDP
CODE        ENDS
            END
```

Voyons un peu comment fonctionne cette routine !... Après chargement de l'adresse du nom de fichier, le fichier correspondant est créé grâce à la fonction 3C de l'interruption 21.

Cela étant, on sélectionne le plan n° 0 de la mémoire EGA et l'on transfère le contenu de ce plan, soit 28 000 octets, dans le fichier nouvellement créé.

Ce plan n° 0 étant transféré, on procède de même pour les 3 autres plans de la mémoire EGA, ce qui donne lieu à un fichier d'image de 112 ko (4 × 28 ko).

Notez l'emploi des mêmes registres AX et DX par la fonction de sélection des plans mémoire et d'écriture dans le fichier. Cela explique pourquoi l'on est obligé de sauvegarder les valeurs de ces deux registres avant l'appel de la fonction d'écriture et de les rétablir aussitôt après.

Programme d'essai en Turbo C

```
main()
{
  int col,x;
  char fichier[13];
  strcpy(fichier,"image.pic");

  screen(16);
  for(x=Ø;x<64Ø;x+=4Ø)              /* Dessin d'une mire 16 couleurs */
  {
    col=x/4Ø;
    xfill(x,Ø,x+39,349,col);
  }
  egasave(fichier);                 /* Sauvegarde image EGA */
}
```

Quitte à sauvegarder une image EGA, autant disposer d'une mire de 16 couleurs couvrant tout l'écran. On est mieux à même de tester le bon déroulement de la sauvegarde d'image.

Comment ?... En effaçant l'écran et en rechargeant l'image que l'on vient de sauvegarder. C'est pourquoi il importe de passer au chapitre suivant.

25

CHARGER UNE IMAGE EGA

A quoi peut bien servir de savoir sauvegarder une image EGA si l'on n'est pas capable de la rappeler. C'est pourquoi nous vous livrons la routine egaload, compagnon indispensable de la routine précédente.

Routine en Assembleur

```
;          ************************************************
;          Routine egaload.asm appelable à partir du Turbo C
;          ************************************************

CODE       SEGMENT BYTE 'CODE'
           ASSUME  CS:CODE
           PUBLIC  _egaload
_egaload   PROC    FAR

           PUSH    BP                ; Sauvegarde pointeur de base.
           MOV     BP,SP             ; Transfert pointeur de pile en BP.
           PUSH    DS                ; Sauvegarde segment de données.
                                     ;
           MOV     DX,(BP+06H)       ; Adresse nom de fichier en DX.
           MOV     AX,3D00H          ; Ouverture du fichier d'image
           INT     21H               ; (fonction 3C, interruption 21).
                                     ;
           MOV     BX,AX             ; Transfert témoin de fichier en BX.
           MOV     CX,28000          ; Nombre d'octet par plan.
```

Routine en Assembleur (suite)

```
                                     ;
            MOV     DX,03C4H         ; Adresse du séquenceur.
            MOV     AX,0102H         ; Sélection du plan n°0
E1:         OUT     DX,AX            ; (code 1, registre 2).
            PUSH    DX               ; Sauvegarde adresse séquenceur.
            PUSH    AX               ; Sauvegarde n° de plan.
                                     ;
            MOV     AX,0A000H        ; Chargement adresse de début
            MOV     DS,AX            ; dans segment de données.
            XOR     DX,DX            ; Raz décalage.
            MOV     AH,3FH           ; Lecture dans fichier d'image
            INT     21H              ; (fonction 3F, interruption 21).
                                     ;
            POP     AX               ; Rétablissement n° de plan.
            POP     DX               ; Rétablissement adresse séquenceur.
            SHL     AH,1             ; Plan suivant (puissance de 2).
            CMP     AH,16            ; A-t-on lu les 4 plans ?
            JL      E1               ; Non, alors on continue.
                                     ;
            MOV     AH,3EH           ; Clotûre du fichier d'image
            INT     21H              ; (fonction 3E, interruption 21).
                                     ;
            POP     DS               ; Rétablissement segment de données.
            POP     BP               ; Rétablissement pointeur de base
            RET                      ; ... et retour à l'envoyeur.

_egaload    ENDP
CODE        ENDS
            END
```

Après chargement en DX de l'adresse du nom de fichier, le fichier correspondant est ouvert grâce à la fonction 3C de l'interruption 21.

Cela étant, on procède ensuite à la sélection en écriture du plan n° 0 de la mémoire EGA et ce, au travers du registre 2 du séquenceur.

La sélection faite, on transfère alors les 28 000 premiers octets du fichier dans ce plan, en débutant à l'adresse A000:0000 que l'on charge dans le couple de registres DS:DX.

Le plan n° 0 étant transféré dans la mémoire EGA, on procède de même pour les 3 autres plans. Notez la numérotation des plans qui s'effectue selon les puissances croissantes de 2 (1, 2, 4, 8).

Le transfert terminé, il ne reste plus qu'à clôturer le fichier à l'aide de la fonction 3E de l'interruption 21 avant de rendre la main au programme appelant.

Programme d'essai en Turbo C

```
main()
{
  int col,x;
  char fichier[13];
  strcpy(fichier,"image.pic");

  screen(16);
  for(x=Ø;x<64Ø;x+=4Ø)              /* Dessin d'une mire 16 couleurs */
  {
    col=x/4Ø;
    xfill(x,Ø,x+39,349,col);
  }
  egasave(fichier);                 /* Sauvegarde image EGA */

  screen(16);                       /* Effacement de l'écran */

  egaload(fichier);                 /* Recharge image EGA */
}
```

Comme vous pouvez le constater, ce programme reprend le dessin de la mire EGA, ainsi que sa sauvegarde et, après avoir effacé l'écran, recharge la mire par l'intermédiaire de la fonction egaload.

26

RELANCER LE SYSTÈME

Pas triste comme méthode de protection !... Vous ne répondez pas correctement à la question que l'on vous pose (mot de passe, touche à enfoncer, etc...) et hop, le système se réinitialise !... C'est en tout cas ce qu'il vous est possible d'accomplir avec la routine reboot *que voici.*

Routine en Assembleur

```
;           ***************************************************
;           Routine reboot.asm appelable à partir du Turbo C
;           ***************************************************

CODE        SEGMENT BYTE 'CODE'
            ASSUME  CS:CODE
            PUBLIC  _reboot
_reboot     PROC    FAR

            INT     19H                 ; Initialisation du système.

_reboot     ENDP
CODE        ENDS
            END
```

Difficile de faire plus court, n'est-ce pas ?... Un appel à l'interruption 19H et hop, le système se réinitialise.

Notez au passage l'absence de RET final. Ce dernier est désormais inutile une fois l'interruption 19 exécutée. Vous pouvez bien sûr le rajouter comme fioriture si vous le souhaitez, mais son usage est superflu.

Programme d'essai en Turbo C

```
main()
{
  printf("Appuyez sur une touche: ");
  getch();
  printf("\nDommage, ce n'était pas la bonne...!");
  reboot();
}
```

Deux mots sur ce très court programme !... Après affichage du message vous invitant à appuyer sur une touche, le programme attend que vous le fassiez.

Selon la touche enfoncée, vous obtenez un message d'erreur, suivi d'une réinitialisation du système, identique à celle que procure l'enfoncement des touches Ctrl - Alt - Del.

Au fait, sur quelle touche fallait-il appuyer ?

27

PROGRAMMER LA SOURIS

Avez-vous déjà essayé de programmer la souris à partir d'un langage compilé ?... Non, n'est-ce pas ?... Sachez que pour y parvenir, il vous faut vous procurer un fichier MOUSE.LIB auprès de Microsoft ou, plus simplement, employer la routine souris que nous vous proposons.

Routine en Assembleur

```
;          ************************************************
;          Routine souris.asm appelable à partir du Turbo C
;          ************************************************

CODE       SEGMENT BYTE 'CODE'
           ASSUME  CS:CODE
           PUBLIC  _souris
_souris    PROC    FAR

           PUSH    BP                  ; Sauvegarde pointeur de base.
           MOV     BP,SP               ; Transfert pointeur de pile en BP.
           PUSH    SI                  ; Sauvegarde index source.
           PUSH    DI                  ; Sauvegarde index destinataire.
                                       ;
           MOV     SI,(BP+Ø6H)         ; Adresse de m1 en SI
           MOV     AX,[SI]             ; et valeur en AX.
           MOV     SI,(BP+Ø8H)         ; Adresse de m2 en SI
           MOV     BX,[SI]             ; et valeur en BX.
```

Routine en Assembleur (suite)

```
        MOV     SI,(BP+ØAH)     ; Adresse de m3 en SI
        MOV     CX,[SI]         ; et valeur en CX.
        MOV     SI,(BP+ØCH)     ; Adresse de m4 en SI
        MOV     DX,[SI]         ; et valeur en DX.
                                ;
        CMP     AX,Ø9H          ; Est-ce la fonction 9 ?
        JE      S1              ; Oui, alors traitement spécial.
        CMP     AX,ØCH          ; Est-ce la fonction 12 ?
        JE      S2              ; Oui, alors traitement spécial.
        CMP     AX,1ØH          ; Est-ce la fonction 16 ?
        JE      S3              ; Oui, alors traitement spécial.
        CMP     AX,14H          ; Est-ce la fonction 2Ø ?
        JE      S4              ; Oui, alors traitement spécial.
        CMP     AX,16H          ; Est-ce la fonction 22 ?
        JE      S1              ; Oui, alors traitement spécial.
        CMP     AX,17H          ; Est-ce la fonction 23 ?
        JNE     S5              ; Non, alors suite sans intérêt.
                                ;
S1:     MOV     DI,DS           ; Traitement fonctions 9, 22 et 23.
        MOV     ES,DI           ; Transfert segment de données
        JMP     SHORT S5        ; dans extra-segment, via DI.
                                ;
S2:     MOV     DI,(BP+Ø4H)     ; Traitement fonction 12.
        MOV     ES,DI           ; Transfert segment de retour
        JMP     SHORT S5        ; dans extra-segment, via DI.
                                ;
S3:     MOV     DI,DX           ; Traitement fonction 16.
        MOV     CX,[DI]         ; Adresse tableau en DI
        MOV     DX,[DI+Ø2H]     ; et chargement valeurs
        MOV     SI,[DI+Ø4H]     ; dans les 4 registres:
        MOV     DI,[DI+Ø6H]     ;  CX=gauche  DX=haut
        JMP     SHORT S5        ;  SI=droite  DI=bas
                                ;
S4:     OR      BX,BX           ; Traitement fonction 2Ø.
        JZ      S2              ; Idem fonction 12 si segment nul.
        MOV     ES,BX           ; Sinon, segment en ES.
                                ;
S5:     PUSH    AX              ; Sauvegarde n° de fonction.
        INT     33H             ; Appel interruption 33.
        POP     DI              ; Rétablissement n° de fonction.
                                ;
        MOV     SI,(BP+Ø6H)     ; Adresse de m1 en SI
        MOV     [SI],AX         ; et retour contenu de AX.
        MOV     SI,(BP+Ø8H)     ; Adresse de m2 en SI
        MOV     [SI],BX         ; et retour contenu de BX.
```

Routine en Assembleur (suite)

```
        CMP     DI,14H          ; Est-ce la fonction 2Ø ?
        JNE     S6              ; Non, alors m2 est correct.
        MOV     [SI],ES         ; Oui, alors m2 = ES.
                                ;
S6:     MOV     SI,(BP+ØAH)     ; Adresse de m3 en SI
        MOV     [SI],CX         ; et retour contenu de CX.
        MOV     SI,(BP+ØCH)     ; Adresse de m4 en SI
        MOV     [SI],DX         ; et retour contenu de DX.
                                ;
        POP     DI              ; Rétablissement index destinataire.
        POP     SI              ; Rétablissement index source.
        POP     BP              ; Rétablissement pointeur de base
        RET                     ; ... et retour à l'envoyeur.

_souris ENDP
CODE    ENDS
        END
```

Le cœur de la routine est constitué par l'appel à l'interruption 33 qui gère l'interface de la souris. Toutefois, avant d'y parvenir, il faut passer au travers de toute une série de tests.

Ces tests visent à déterminer, après qu'aient été chargés les 4 paramètres m1 à m4 dans les registres AX à DX, si la fonction sélectionnée (de 0 à 30) ne nécessite pas un traitement spécial. C'est notamment le cas des fonctions 9, 12, 16, 20, 22 et 23.

Ce traitement éventuel accompli, on sauvegarde le numéro de la fonction de façon à conserver l'information au retour de l'interruption 33. En effet, selon la fonction, le contenu du registre AX peut être écrasé.

Notez que l'on réexpédie au retour de l'interruption 33 le contenu des registres AX à DX vers les 4 paramètres m1 à m4, à l'exception toutefois du registre BX dans la fonction 20 qui est remplacé par l'extra-segment. C'est d'ailleurs pour pouvoir effectuer ce test que l'on a sauvegardé précédemment le numéro de la fonction.

Important : *Pour adapter cette routine à d'autres langages que le Turbo C, seules les 8 lignes de passage des paramètres, et éventuellement le RET final, sont à modifier. En aucun cas, il ne faut toucher au BP + 04H de traitement de la fonction 12.*

Programme d'essai en Turbo C

```
main()
{
  int m1,m2,m3,m4;
  m1=Ø;
  souris(&m1,&m2,&m3,&m4);
  if (m1==Ø)
  {
    printf("Interface souris non installée...");
    exit(Ø);
  }
  else
  {
    m1=1;
    souris(&m1,&m2,&m3,&m4);
    while (m2!=3)
    {
      m1=3;
      souris(&m1,&m2,&m3,&m4);
    }
  }
}
```

Après déclaration des 4 paramètres m1 à m4, on interroge à l'aide de la fonction 0 (m1 = 0) l'interruption 33 pour savoir si l'interface de la souris est installée. Pour information, la fonction 0 renvoie la valeur -1 dans m1 lorsque l'interface est installée, 0 dans le cas contraire.

Cela étant, et l'interface étant installée, on rend visible la souris à l'aide de la fonction 1 et on récupère l'état de ses boutons, ainsi que sa position avec la fonction 3. Notez qu'il faut appuyer simultanément sur les 2 boutons pour sortir de cette boucle de lecture.

28

PARAMÉTRER LE PORT SÉRIE

Bien que la plupart des langages disposent d'instructions de paramétrage du port série (OPENCOM des Basics, ENABIN du Pascal, Int du Turbo-Pascal, bios_serial_com du C et du Quick C, bioscom du Turbo C...), il en est un qui en est démuni : le Fortran. C'est pour pallier à cette carence que nous avons dû développer la routine opencom que voici.

Routine en Assembleur

```
;           ***************************************************
;           Routine opencom.asm appelable à partir du Turbo C
;           ***************************************************


CODE        SEGMENT BYTE 'CODE'
            ASSUME  CS:CODE
            PUBLIC  _opencom
_opencom    PROC    FAR

            PUSH    BP              ; Sauvegarde pointeur de base.
            MOV     BP,SP           ; Transfert du pointeur de pile en BP.
                                    ;
            MOV     DX,(BP+Ø6H)     ; Chargement du n° de port.
            DEC     DX              ; On décrémente DX (Ø=COM1 1=COM2).
                                    ;
            MOV     BX,(BP+Ø8H)     ; Chargement de la vitesse
            MOV     AL,Ø            ; (vitesse: Ø=11Ø 7=96ØØ).
```

Routine en Assembleur (suite)

```
                                        ;
01:        CMP     BX,11Ø               ; Vitesse <= 11Ø Baud ?
           JLE     02                   ; Oui, alors on saute en 02.
           INC     AL                   ; Non, alors on incrémente AL,
           SHR     BX,1                 ; on divise par 2 la vitesse
           JMP     SHORT 01             ; et on recommence le test.
02:        MOV     CL,3                 ; Rotation de 3 bits vers la droite.
           ROR     AL,CL                ; Vitesse: bits 5, 6 et 7 de AL.
                                        ;
           MOV     BX,(BP+ØAH)          ; Chargement de la parité
           SHL     BL,CL                ; (Ø=sans 1=impaire 2=paire).
           OR      AL,BL                ; Parité: bits 3 et 4 de AL.
                                        ;
           MOV     BX,(BP+ØCH)          ; Chargement nombre de bits
           AND     BL,ØØØØØØ1ØB         ; de fin (1bit=Ø 2bits=1) que
           SHL     BL,1                 ; l'on place après masquage et
           OR      AL,BL                ; décalage dans le bit 2 de AL.
                                        ;
           MOV     BX,(BP+ØEH)          ; Chargement nombre de bits
           SUB     BL,5                 ; de données (7bits=1Ø 8bits=11).
           OR      AL,BL                ; Bits de données: bits Ø et 1 de AL.
                                        ;
           MOV     AH,Ø                 ; Envoi du tout au port série
           INT     14H                  ; (fonction Ø interruption 14H).
                                        ;
           POP     BP                   ; Rétablissement pointeur de base
           RET                          ; ... et retour à l'envoyeur.

_opencom   ENDP
CODE       ENDS
           END
```

Comment fonctionne cette routine opencom ?... Après avoir chargé le numéro de port, on le décrémente de façon à l'adapter aux spécifications de l'interruption 14 (0 = COM1, 1 = COM2).

Cette opération accomplie, on charge alors la vitesse (110 à 9600 bauds) qu'on traite par divisions successives de façon à obtenir son n° de code (0 = 110 bauds à 7 = 9600 bauds), code que l'on positionne dans les bits 5 à 7 du registre AL.

Vient ensuite la parité (0 = sans, 1 = impaire, 2 = paire) dont on passe le code, à un décalage près, dans les bits 3 et 4 de AL, puis le nombre de bits de fin dans le bit 2 et, pour terminer, le nombre de bits de données dans les bits 0 et 1.

Le registre AL étant alors plein, il ne reste plus qu'à expédier son contenu au port série par l'intermédiaire de l'interruption 14H du Bios.

Important : *L'ensemble de cette routine a été développé autour de l'interruption 14. Sachez que cette solution est nettement préférable à celle qui consiste à travailler directement dans les registres du port série. Nous avons « grillé » plusieurs cartes série pour ne pas avoir compris cela.*

Programme d'essai en Turbo C

```
main()
{
  opencom(1,9600,2,2,8);
}
```

Que dire de ce programme d'essai si ce n'est qu'il configure le port n° 1 avec une vitesse de 9600 Bauds, parité paire en service, deux bits de fin et 8 bits de données.

29

PILOTER UNE TABLETTE GRAPHIQUE

Les tablettes graphiques étant dépourvues pour la plupart de logiciel pilote (que d'aucuns nomment « driver »...), il importe au développeur d'un programme de les écrire lui-même. Or, le savoir-faire en ce domaine n'est (à notre connaissance...) nulle part consigné par écrit. Désormais, c'est chose faite.

Vu le grand nombre de tablette graphiques sur le marché, il est bien évident que le présent chapitre ne peut en aucun cas passer en revue de manière exhaustive chacun des modèles existant. Tout au plus peut-il vous proposer un exemple concret que les auteurs ont réalisé à partir du modèle de tablette en leur possession, à savoir la Summagraphics MM1812.

Cela dit, il est relativement facile de passer d'un modèle à un autre dans la mesure où l'on a bien assimilé les différentes phases de réalisation d'un logiciel pilote. Seuls changent quelques points de détails...

Routine en Assembleur

```
;          ***************************************************
;          Routine report.asm appelable à partir du Turbo C
;          ***************************************************
;
;          Pour obtenir un fonctionnement correct de cette routine,
;          il convient de positionner comme suit les micro-interrupteurs
;          de la tablette graphique SUMMAGRAPHICS MM1812:
;
;          SW1-1:  On    \
;          SW1-2:  Off     Vitesse de transmission de 9600 Bauds
;          SW1-3:  Off   /
;          SW1-4:  On    \
;          SW1-5:  On    / Parité paire.
;          SW1-6:  On      2 bits de fin.
;          SW1-7:  Off     Pas d'écho.
;          SW1-8:  On      Pas d'arrêt de transmission.
;
;          SW2-1:  Off   \
;          SW2-2:  On    / Résolution de 1000 lpi (lignes/pouce).
;          SW2-3:  Off     Sans signification (option ASCII).
;          SW2-4:  On      Format binaire.
;          SW2-5:  On      Sans signification (option ASCII).
;          SW2-6:  On      Sans signification (option ASCII).
;          SW2-7:  On    \
;          SW2-8:  On    / Mode permanent.
;
;          SW3-1:  Off     Type UIOF.
;          SW3-2:  On      Sans signification (option ASCII).
;          SW3-3:  Off     Identificateur de tablette à 0.
;          SW3-4:  On    \
;          SW3-5:  On    / Non utilisés.
;          SW3-6:  On    \
;          SW3-7:  On      Cadence de 80 rapports/s.
;          SW3-8:  On    /

           CODE    SEGMENT BYTE 'CODE'
           ASSUME  CS:CODE
           PUBLIC  _report
_report    PROC    FAR

           PUSH    BP              ; Sauvegarde pointeur de base.
           MOV     BP,SP           ; Transfert du pointeur de pile en BP.
           PUSH    SI              ; Sauvegarde index source.
                                   ;
           MOV     DX,0            ; Chargement n° de port (0=COM1 1=COM2).
```

Routine en Assembleur (suite)

```
                                   ;
R1:     MOV     AH,2               ; Réception du 1er octet
        INT     14H                ; (fonction 2 interruption 14H).
        MOV     BL,AL              ; Sauvegarde en BL.
        AND     AL,01000000B       ; Est-ce l'octet de synchro ?
        JZ      R1                 ; Non, alors on recommence.
        AND     BL,00000001B       ; Oui, alors on récupère le
        MOV     CL,3               ; bit de proximité qui passe
        ROR     BL,CL              ; de la position n°0 à la n°5.
                                   ;
        MOV     AH,2               ; Réception du 2e octet
        INT     14H                ; (fonction 2 interruption 14H).
        AND     AL,00001111B       ; Sélection des 4 bits utiles.
        OR      BL,AL              ; Incorporation du bit de proximité
        XOR     BH,BH              ; à l'état des boutons (+32 si hors).
        MOV     SI,(BP+0AH)        ; Chargement adresse de Bouton en SI
        MOV     [SI],BX            ; et renvoi contenu de BX en Bouton.
                                   ;
        MOV     AH,2               ; Réception du 3e octet
        INT     14H                ; (fonction 2 interruption 14H).
        AND     AL,00111111B       ; Sélection des 6 bits utiles
        MOV     BL,AL              ; et leur transfert en BL.
                                   ;
        MOV     AH,2               ; Réception du 4e octet
        INT     14H                ; (fonction 2 interruption 14H).
        AND     AL,00111111B       ; Sélection des 6 bits utiles
        ROR     AL,1               ; que l'on positionne par 2
        ROR     AL,1               ; rotations à droite de AL.
        MOV     CL,AL              ; Sauvegarde de AL en CL.
        AND     AL,11000000B       ; Sélection des bits 6 et 7
        OR      BL,AL              ; que l'on transfère en BL.
        XOR     BH,BH              ; Remise à zéro de AH.
        AND     CL,00001111B       ; Sélection des bits 8 à 11
        OR      BH,CL              ; que l'on transfère en BH.
                                   ;
        MOV     AH,2               ; Réception du 5e octet
        INT     14H                ; (fonction 2 interruption 14H).
        AND     AL,00000111B       ; Sélection des bits 12 à 14
        MOV     CL,4               ; que l'on positionne par 4
        ROR     AL,CL              ; rotations à droite de AL.
        OR      BH,AL              ; Transfert de AL en BH.
        MOV     SI,(BP+06H)        ; Chargement adresse de Xt en SI
        MOV     [SI],BX            ; et renvoi contenu de BX en Xt.
```

Routine en Assembleur (suite)

```
                                  ;
         MOV    AH,2              ; Réception du 6e octet
         INT    14H               ; (fonction 2 interruption 14H).
         AND    AL,ØØ111111B      ; Sélection des 6 bits utiles
         MOV    BL,AL             ; et leur transfert en BL.
                                  ;
         MOV    AH,2              ; Réception du 7e octet
         INT    14H               ; (fonction 2 interruption 14H).
         AND    AL,ØØ111111B      ; Sélection des 6 bits utiles
         ROR    AL,1              ; que l'on positionne par 2
         ROR    AL,1              ; rotations à droite de AL.
         MOV    CL,AL             ; Sauvegarde de AL en CL.
         AND    AL,11ØØØØØØB      ; Sélection des bits 6 et 7
         OR     BL,AL             ; que l'on transfère en BL.
         XOR    BH,BH             ; Remise à zéro de BH.
         AND    CL,ØØØØ1111B      ; Sélection des bits 8 à 11
         OR     BH,CL             ; que l'on transfère en BH.
                                  ;
         MOV    AH,2              ; Réception du 8e octet
         INT    14H               ; (fonction 2 interruption 14H).
         AND    AL,ØØØØØ111B      ; Sélection des bits 12 à 14
         MOV    CL,4              ; que l'on positionne par 4
         ROR    AL,CL             ; rotations à droite de AL.
         OR     BH,AL             ; Transfert de AL en BH.
                                  ;
         MOV    SI,(BP+Ø8H)       ; Chargement adresse de Yt en SI
         MOV    [SI],BX           ; et renvoi contenu de BX en Yt.
                                  ;
         POP    SI                ; Récupération index source.
         POP    BP                ; Récupération pointeur de base
         RET                      ; ... et retour à l'envoyeur.

_report  ENDP
CODE     ENDS
         END
```

Ayant sélectionné pour des raisons de vitesse le format binaire plutôt que le format ASCII pour la transmission des informations (coordonnées X, Y du curseur et état des boutons...) entre la tablette et l'ordinateur, il nous faut nous reporter à la documentation de celle-ci pour prendre connaissance du nombre d'octets que nécessite la transmission complète d'un rapport sous ce format binaire.

Or, de par sa grande taille (format A3), la tablette Summagraphics MM1812 gère des nombres sur 16 bits auquel est associé un bit de signe. Dans ces conditions, la séquence de transmission est plus longue que la normale qui, de 5 octets qu'elle est sur les tablettes au format A4, passe à 8 octets sur ce modèle un peu plus grand.

Voici d'ailleurs, extrait de la documentation (indispensable...) de la tablette, le contenu détaillé des 8 octets de la séquence binaire de transmission :

Bit n°								**Séquence de transmission**
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
P	Sync	0	0	Mb	Ma	T	Prox	1er octet
P	0	0	0	Fd	Fc	Fb	Fa	2ème octet
P	0	X5	X4	X3	X2	X1	X0	3ème octet
P	0	X11	X10	X9	X8	X7	X6	4ème octet
P	0	0	Sx	X15	X14	X13	X12	5ème octet
P	0	Y5	Y4	Y3	Y2	Y1	Y0	6ème octet
P	0	Y11	Y10	Y9	Y8	Y7	Y6	7ème octet
P	0	0	Sy	Y15	Y14	Y13	Y12	8ème octet

Vous noterez qu'hormis les identificateurs de tablette (T, Ma et Mb) qui ne nous sont d'aucune utilité dans le moment présent, il va nous falloir extraire de ces 8 octets afin de les mettre en forme les informations :

Sync - de synchronisation de la séquence
P - de parité (il s'agit d'une 2nd parité)
Prox - de proximité
Fa - Fd - d'état des boutons
X - Y - de position du curseur

Ceci dit, vous pouvez constater que la routine débute, après les habituelles sauvegarde, par la recherche du bit de synchronisation (c'est la raison d'être du masque 01000000...).

La séquence étant synchronisée, on récupère alors le bit de proximité que l'on décale de 5 bits vers la gauche (2^5 = 32) de façon à l'incorporer à la variable Bouton. Notez qu'au lieu de la pile, c'est le registre BL qui assure la sauvegarde de cette valeur.

Cela étant, on réceptionne les 3ème, 4ème et 5ème octets contenant la coordonnée X de position du curseur. Après avoir sélectionné les 6 bits utiles de poids faible du 3ème octet, on décale de 2 bits vers la droite le 4ème octet de façon à placer au bon endroit les bits n° 6 et 7.

Cette étape accomplie, on récupère les bits n° 8 à 11 et, après chargement du 5ème octet et son décalage de 4 bits vers la droite, on finit de positionner correctement les bits n° 12 à 14. Notez à ce propos que l'on ne traite ni le bit n° 15, ni le bit de signe.

La coordonnée X étant expédiée vers la variable du même nom, on procède alors au traitement (identique...) de la coordonnée Y avant de l'expédier à son tour vers le programme d'appel.

Programme d'essai en Turbo C

```
main()
{
  int xt,yt,bouton;
  opencom(1,9600,2,2,8);
  xt=yt=bouton=0;
  while(bouton<32)
  {
    report(&xt,&yt,&bouton);
    locate(25,1);
    printf("xt=%5d yt=%5d bouton=%2d",xt,yt,bouton);
  }
}
```

Après une remise à zéro systématique des variable Xt, Yt et Bouton, la fonction externe report ramène de la tablette les nouvelles valeurs de position du curseur et d'état des boutons (à raison de 80 rapports/seconde...), valeurs que l'on s'empresse d'afficher de façon à s'assurer que la routine de pilotage fonctionne correctement.

30

RECOPIER UN ÉCRAN EGA

L'avènement du mode graphique EGA ne fut pas suivi par la mise à niveau du fichier GRAPHICS de recopie d'écran. Il s'en suit que pour recopier un écran EGA, vous n'avez d'autre alternative que de vous procurer un logiciel spécialisé. Désormais ce n'est plus le cas, le savoir-faire en ce domaine vous étant dévoilé au travers de la routine egaprtsc que voici.

Vu le très grand nombre de modèles d'imprimantes existant sur le marché, il ne peut être question dans le présent chapitre de passer en revue les différences qui les distinguent les unes des autres sur le plan programmation. Comme nous disposions d'une imprimante EPSON EX800, le choix fut vite fait...

Ceci dit il est relativement aisé de passer d'un modèle à un autre, dès lors que l'on a bien assimilé les étapes de réalisation d'un fichier .COM résidant de recopie d'écran. Seuls changent quelques codes d'imprimante...

Routine en Assembleur

```
;          ***********************************************
;          Routine egaprtsc.asm écrite pour fichier .COM
;          ***********************************************
;
;
;          Avertissement: Cette routine de recopie d'écrans EGA est
;                         conçue pour piloter une imprimante EPSON EX800.
;                         Elle est susceptible de modifications avec tout
;                         autre modèle d'imprimante.

CODE       SEGMENT
           ASSUME  CS:CODE
           ASSUME  DS:CODE

           ORG     100H

egaprtsc:  JMP     test

exit       LABEL   DWORD             ; Etiquette du saut long.
old_off    DW      ?                 ; Ancien vecteur
old_seg    DW      ?                 ; interruption 5H.
new_seg    DW      ?                 ; Segment nouveau vecteur 5H.

texte1     DB      'Recopie d',27h,'écran installée...',0dh,0ah,24h
texte2     DB      'Recopie d',27h,'écran déjà résidante...',0dh,0ah,24h

entree:    PUSH    AX                ; Sauvegarde ...
           PUSH    BX                ; registres AX et BX,
           PUSH    CX                ; registre CX,
           PUSH    DX                ; registre DX,
           PUSH    SI                ; index source,
           PUSH    DI                ; index destination,
           PUSH    DS                ; segment de données,
           PUSH    ES                ; extra segment et
           PUSH    BP                ; pointeur de base.
                                     ;
           MOV     AH,0FH            ; Lecture du
           INT     10H               ; mode vidéo.
           CMP     AL,0DH            ; Est-ce un mode EGA ?
           JGE     R1                ; Oui, alors on recopie.
                                     ;
           POP     BP                ; Non, alors rétablissement...
           POP     ES                ; pointeur de base,
           POP     DS                ; segments ES et DS,
           POP     DI                ; index destination,
           POP     SI                ; index source,
```

Routine en Assembleur (suite)

```
         POP    DX               ; registre DX,
         POP    CX               ; registre CX,
         POP    BX               ; registre BX,
         POP    AX               ; et registre AX.
         JMP    CS:exit          ; Saut long vers recopie texte.
                                 ;
R1:      XOR    AX,AX            ; Revectorisation de
         MOV    ES,AX            ; l'interruption 5H
         MOV    ES:14H,OFFSET fin; sur l'IRET de fin.
                                 ;
         MOV    AX,0A000H        ; Chargement adresse de début
         MOV    ES,AX            ; dans extra segment via AX.
                                 ;
         XOR    DX,DX            ; Imprimante n°0.
         MOV    AX,001BH         ; Initialisation
         INT    17H              ; de l'imprimante.
         MOV    AX,0040H         ; Envoi des codes
         INT    17H              ; imprimante Esc @.
                                 ;
         MOV    AX,001BH         ; Sélection
         INT    17H              ; interlignage
         MOV    AX,0041H         ; vertical de
         INT    17H              ; 8/72e pouce.
         MOV    AX,0008H         ; Envoi des codes
         INT    17H              ; imprimante Esc A n.
                                 ;
         MOV    DI,79            ; Début colonne n°79.
R2:      MOV    SI,0             ; Début ligne n°0.
         XOR    BP,BP            ; (raz SI et BP).
                                 ;
         MOV    AX,001BH         ; Impression en
         INT    17H              ; mode graphique
         MOV    AX,004BH         ; simple densité
         INT    17H              ; (60 points/pouce).
         MOV    AX,005EH         ; Indication du
         INT    17H              ; nombre d'octets
         MOV    AX,0001H         ; à imprimer (350).
         INT    17H              ; Codes Esc K n1 n2.
                                 ;
R3:      XOR    BH,BH            ; Raz octet d'image.
         MOV    DX,03CEH         ; Port du contrôleur graphique.
         MOV    AX,0004H         ; Sélection du plan n°0
R4:      OUT    DX,AX            ; (code 0 registre 4).
         MOV    BL,ES:[BP+DI]    ; Chargement octet du plan et
         OR     BH,BL            ; mixage avec octet d'image.
         INC    AH               ; Sélection du plan suivant.
         CMP    AH,4             ; A-t-on lu les 4 plans ?
         JL     R4               ; Non, alors on recommence.
```

Routine en Assembleur (suite)

```
                                        ;
         MOV     CL,8                   ; Permutation
R5:      SHR     BH,1                   ; des 8 bits de
         RCL     AL,1                   ; l'octet d'image
         DEC     CL                     ; lors du transfert
         JNZ     R5                     ; de BH en AL.
                                        ;
         XOR     DX,DX                  ; Imprimante n°Ø.
         XOR     AH,AH                  ; Envoi octet d'image
         INT     17H                    ; à l'imprimante.
                                        ;
         ADD     BP,8Ø                  ; Incrémentation adresse et
         INC     SI                     ; numéro de ligne suivante.
         CMP     SI,35Ø                 ; A-t-on imprimé 35Ø lignes ?
         JL      R3                     ; Non, alors on recommence.
                                        ;
         MOV     AX,ØØØAH               ; Envoi saut de ligne
         INT     17H                    ; à l'imprimante.
                                        ;
         DEC     DI                     ; A-t-on imprimé 8Ø colonnes ?
         JGE     R2                     ; Non, alors on recommence.
                                        ;
         MOV     AX,ØØ1BH               ; Initialisation
         INT     17H                    ; de l'imprimante.
         MOV     AX,ØØ4ØH               ; Envoi des codes
         INT     17H                    ; imprimante Esc @.
                                        ;
         XOR     AX,AX                  ; Suppression pointage
         MOV     ES,AX                  ; sur l'IRET de fin et
         MOV     AX,OFFSET entree       ; rétablissement vecteur
         MOV     ES:14H,AX              ; interruption 5H.
                                        ;
         POP     BP                     ; Non, alors rétablissement...
         POP     ES                     ; pointeur de base,
         POP     DS                     ; segments ES et DS,
         POP     DI                     ; index destination,
         POP     SI                     ; index source,
         POP     DX                     ; registre DX,
         POP     CX                     ; registre CX,
         POP     BX                     ; registre BX,
         POP     AX                     ; et registre AX.
fin:     IRET                           ; Retour au DOS.
```

Routine en Assembleur (suite)

```
;---------------------------------
;        Partie non résidante
;---------------------------------

test:     XOR    AX,AX              ; Extra-segment
          MOV    ES,AX              ; égal à zéro.
          MOV    AX,ES:16H          ; Segment vecteur 5H
          MOV    ES,AX              ; correspond-il avec
          CMP    AX,ES:107H         ; new_seg déjà installé ?
          JNE    install            ; Non, alors on installe.
                                    ;
          MOV    DX,OFFSET texte2   ; Recopie d'écran
          MOV    AH,09H             ; déjà résidante.
          INT    21H                ; Affichage message
          INT    20H                ; et retour au DOS.
                                    ;
install:  XOR    AX,AX              ; Extra-segment
          MOV    ES,AX              ; égal à zéro.
          MOV    AX,ES:14H          ; Sauvegarde ancien vecteur
          MOV    old_off,AX         ; de l'interruption 5H
          MOV    AX,ES:16H          ; décalage en old_off
          MOV    old_seg,AX         ; segment en old_seg.
                                    ;
          MOV    AX,OFFSET entree   ; Ecriture nouveau vecteur
          MOV    ES:14H,AX          ; de l'interruption 5H et
          MOV    ES:16H,CS          ; signature installation
          MOV    new_seg,CS         ; de la recopie en new_seg.
                                    ;
          MOV    DX,OFFSET texte1   ; Affichage message
          MOV    AH,09H             ; recopie installée
          INT    21H                ; (fonction 09).
                                    ;
          MOV    DX,OFFSET fin      ; Recopie d'écran EGA
          INT    27H                ; désormais résidante.

CODE      ENDS
          END    egaprtsc
```

Le fichier .COM de recopie d'écran se devant d'être résidant de façon à être appelable par l'interruption 5 (enfoncement des touches Shift-Prtsc), son écriture est sensiblement différente de celle employée pour les autres fichiers .COM de cet ouvrage.

Tout d'abord comment rend-on un fichier .COM résidant du DOS ?... En oubliant de l'effacer après son installation, ce que réalise fort bien l'interruption 27.

Notez que ce fichier se compose de 2 parties, une partie résidante qui correspond à la recopie d'écran et une partie non résidante d'installation.

En ce qui concerne l'installation, la routine commence par vérifier si la valeur du segment du vecteur d'interruption 5H (située à l'adresse 16H du segment 0) est identique à celle mémorisée dans la variable new_seg. Si c'est le cas, c'est la preuve que la recopie d'écran est déjà installée.

Si, par contre ce n'est pas le cas, on procède à l'installation en débutant par la sauvegarde de l'ancienne valeur du vecteur 5H. Pour mémoire, elle correspond au point d'entrée de la routine Prtsc de recopie des écrans textes.

Cette sauvegarde accomplie, on peut installer le nouveau vecteur de l'interruption 5H qui correspond au point d'entrée (étiquette du même nom...) de la routine de recopie et, après avoir signalé l'installation, retourner au DOS via l'interruption 27.

Côté recopie d'écran, on débute par la sauvegarde de l'ensemble des registres avant d'effectuer le test du mode vidéo utilisé. Si le test décèle que l'on n'est pas en mode EGA, on s'empresse de rétablir l'ensemble des registres et on effectue un saut (long...) vers la routine de recopie des écrans textes.

S'il s'avère qu'on est en mode EGA, on revectorise aussitôt le vecteur 5H de façon à le faire pointer sur un IRET pendant toute la durée de la recopie. Pourquoi cela ?... Pour protéger la routine contre la manœuvre intempestive des touches Shift-Prtsc pendant qu'une recopie est en cours.

Après cela, on initialise l'imprimante et on sélectionne l'interlignage de 8/72e pouce, ce qui correspond à l'emploi de 8 aiguilles sur les 9 dont dispose la machine. Notez au passage que l'envoi des codes à l'imprimante s'effectue par l'intermédiaire de l'interruption 17H.

On débute alors par le dessin de la dernière colonne de l'écran (l'impression sur papier étant orientée de 90 degrés par rapport à l'écran du moniteur...) et on spécifie à l'imprimante le nombre d'octets à imprimer par ligne, ainsi que la densité d'impression (60 points/pouce dans notre cas...).

On procède alors à la lecture et à la superposition de chaque octet d'image des 4 plans, non sans oublier de permuter l'ordre des bits de l'octet résultant de façon à les faire correspondre aux 8 aiguilles de la tête d'impression.

Cela étant, on expédie les 350 octets d'une ligne à l'imprimante et, après l'envoi d'un saut de ligne, on récidive pour les 79 autres colonnes. Notez que l'on emploie l'index source SI comme compteur de lignes et le pointeur de base

comme adresse de début (colonne 0) de chaque ligne, l'adresse d'une ligne s'obtenant par addition de 8C octets à celle de la ligne qui précède.

L'impression étant finie, on réinitialise l'imprimante, on rétablit le vecteur 5H et on termine en rétablissant l'ensemble des registres.

Création d'un fichier .COM

Destinée à devenir résidante sous la forme d'un fichier .COM, cette routine débute par l'inévitable ORG 100H.

Notez l'absence de déclaration de type public et de procédure FAR, avec ce que cela implique de transfert du pointeur de pile en BP. Ici, tout cela est inutile !... Il suffit de déclarer le segment de code, un point c'est (presque...) tout !

Notez également l'emploi d'un JMP pour sauter par dessus la zone des variables de la portion résidante et la zone des messages d'erreur de la portion non résidante. A ce propos, il est préférable de ne pas rendre résidant les messages d'erreur puisqu'ils ne servent qu'à l'installation.

Ceci dit, il reste à obtenir un fichier .COM à partir du code source de cette routine. Pour ce faire, il vous faut taper la suite d'ordres que voici :

```
>MASM EGAPRTSC;

>LINK EGAPRTSC;

>EXE2BIN EGAPRTSC EGAPRTSC.COM
```

Si tout se passe bien, vous devriez aboutir à un fichier EGAPRTSC.COM de 336 octets. Pas mal, non ?

Programme d'essai en Turbo C

Pour ceux qui ne disposerait d'aucune image EGA, voici de quoi réaliser une mire de quadrillage de maille quasi-carrée. Elle vous permettra d'apprécier la distorsion de reproduction qu'introduit (ou non...) votre imprimante.

```
C:\TURBOC>exit
#include <graphics.h>

main()
{
  int x,y,driver=3,mode=1;
  initgraph(&driver,&mode,"");
  setcolor(13);
  rectangle(0,0,639,349);

  for(x=32;x<640;x+=32)
    line(x,0,x,349);
  for(y=25;y<350;y+=25)
    line(0,y,639,y);

  getch();
}
```

Ecrit en Turbo C 1.5, ce petit programme fait appel aux fonctions de la librairie graphique de ce langage pour passer en mode EGA, sélectionner la couleur d'encre et dessiner cadre et quadrillage.

Exécutez-le et appuyez alors sur les touches Shift-Prtsc pour obtenir une recopie d'écran, non sans avoir au préalable installé le fichier résidant EGAPRTSC.COM.

LE CODE ASCII

Poids	Caractère ou code	
000	(nul)	NUL
001	☺	SOH
002	●	STX
003	♥	ETX
004	♦	EOT
005	♣	ENQ
006	♠	ACK
007	("bip")	BEL
008	(Espace arrière)	BS
009	(Tabulation)	HT
010	(Ligne suivante)	LF
011	(Home)	VT
012	(Page suivante)	FF
013	(Retour chariot)	CR
014	♫	SO
015	☼	SI
016	►	DLE
017	◄	DC1
018	↕	DC2
019	‼	DC3
020	π	DC4
021	§	NAK
022	▬	SYN
023	↨	ETB
024	↑	CAN
025	↓	EM

Poids	Caractère ou code	
026	→	SUB
027	←	ESC
028	(Curseur à droite)	FS
029	(Curseur à gauche)	GS
030	(Curseur vers le haut)	RS
031	(Curseur vers le bas)	US
032	(Espace)	
033	!	
034	"	
035	#	
036	$	
037	%	
038	&	
039	'	
040	(	
041	)	
042	*	
043	+	
044	,	
045	-	
046	.	
047	/	
048	0	
049	1	
050	2	
051	3	

Poids	Caractère ou code
052	4
053	5
054	6
055	7
056	8
057	9
058	:
059	;
060	<
061	=
062	>
063	?
064	@
065	A
066	B
067	C
068	D
069	E
070	F
071	G
072	H
073	I
074	J
075	K
076	L
077	M

Le code ASCII (suite)

Poids	Caractère ou code
078	N
079	O
080	P
081	Q
082	R
083	S
084	T
085	U
086	V
087	W
088	X
089	Y
090	Z
091	[
092	\
093	]
094	^
095	_
096	'
097	a
098	b
099	c
100	d
101	e
102	f
103	g
104	h
105	i
106	j
107	k
108	l
109	m
110	n
111	o
112	p
113	q
114	r
115	s
116	t
117	u
118	v
119	w
120	x
121	y
122	z
123	{
124	¦
125	}
126	~
127	⌂
128	Ç
129	ü
130	é
131	â
132	ä
133	à
134	å
135	ç
136	ê
137	ë

Poids	Caractère ou code
138	è
139	ï
140	î
141	ì
142	Ä
143	Å
144	É
145	æ
146	Æ
147	ô
148	ö
149	ò
150	û
151	ù
152	ÿ
153	Ö
154	Ü
155	¢
156	£
157	¥
158	Pt
159	ƒ
160	á
161	í
162	ó
163	ú
164	ñ
165	Ñ
166	ª
167	º
168	¿
169	⌐
170	¬
171	1/2
172	1/4
173	¡
174	«
175	»
176	░
177	▒
178	▓
179	│
180	┤
181	╡
182	╢
183	╖
184	╕
185	╣
186	║
187	╗
188	╝
189	╜
190	╛
191	┐
192	└
193	┴
194	┬
195	├
196	─
197	┼

Poids	Caractère ou code
198	╞
199	╟
200	╚
201	╔
202	╩
203	╦
204	╠
205	═
206	╬
207	╧
208	╨
209	╤
210	╥
211	╙
212	╘
213	╒
214	╓
215	╫
216	╪
217	┘
218	┌
219	█
220	▄
221	▌
222	▐
223	▀
224	α
225	β
226	Γ
227	π
228	Σ
229	σ
230	μ
231	τ
232	Φ
233	Θ
234	Ω
235	δ
236	∞
237	∅
238	∈
239	∩
240	≡
241	±
242	≥
243	≤
244	⌠
245	⌡
246	÷
247	≈
248	°
249	•
250	·
251	√
252	n
253	²
254	■
255	(Blanc "FF")

TABLE DES MATIÈRES

(*) XOR : mode qui permet de déplacer des entités (poins, traits, zones coloriées, icônes, etc.) sur l'écran sans altérer le fond d'image.

Service lecteurs

(à retourner à Éditions Radio, 189, rue Saint-Jacques, 75005 Paris)

Pour nous permettre de vous proposer des ouvrages toujours meilleurs, nous souhaiterions recevoir vos critiques, appréciations et suggestions sur le présent livre :

____

____

____

____

Quels sont les ouvrages (thème, sujet, niveau) que vous souhaiteriez voir publier par notre société ?

____

____

____

____

Nous vous remercions de votre confiance et de votre coopération.

Éditions Radio

Je désire recevoir gratuitement et sans engagement (mettre une croix dans la case) :

☐ Votre catalogue général (Electronique professionnelle et grand public, Informatique, Hi-Fi, Vidéo)

Nom : ____________________ Prénom : ____________________

Adresse : ____________________________________

____

Secteur d'activité et fonction : __________________________

____

CENTRES D'INTÉRÊTS

☐ Electronique professionnelle
☐ Electronique de loisirs
☐ Vidéo
☐ Hifi, CB...
☐ Micro-informatique professionnelle
☐ Micro-informatique de loisirs
☐ Autres : ....
____

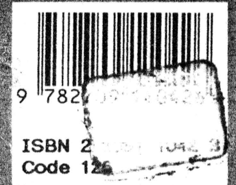
30 ROUTINES ASSEMBLEUR

Les langages de programmation étant incomplets, voire dépourvus — pour certains — des instructions que vous jugez indispensables, il importe que vous sachiez les enrichir en leur intégrant, sous la forme de routines écrites en Assembleur, les éléments de programmation qui leur manquent.

Ce livre vous propose, à vous qui avez appris à manipuler l'Assembleur, des routines toutes prêtes à être intégrées dans votre langage de programmation favori, qu'il s'agisse du :
Basica - GW Basic - Basic compilé - Quick Basic - Turbo Basic - Pascal - Turbo Pascal - Fortran - C Microsoft - Quick C - Turbo C.

Ces 30 routines constituent la toile de fond de bien des programmes professionnels, tant en ce qui concerne le graphisme à très haute résolution, que le pilotage des périphériques d'entrée-sortie (souris, tablette graphique, imprimante, etc.) :

— Vider la mémoire-tampon du clavier
— Obtenir le code ASCII d'une touche
— Activer la touche Caps Lock
— Désactiver la touche Caps Lock
— Activer la touche Num Lock
— Désactiver la touche Num Lock
— Lire l'état des touches spéciales
— Positionner le curseur
— Afficher 43 lignes de texte
— Chronométrer au 1/18e de seconde
— Lire l'heure
— Lire la date
— Jouer une note de musique
— Passer en mode graphique EGA
— Délimiter une fenêtre graphique
— Tirer un trait en XOR
— Déplacer un réticule
— Colorier un rectangle en XOR
— Afficher une icône
— Protéger un fichier contre l'effacement
— Déverrouiller un fichier protégé
— Cacher un fichier
— Rendre visible un fichier caché
— Sauvegarder une image EGA
— Charger une image EGA
— Relancer le système
— Programmer la souris
— Paramétrer le port série
— Piloter une tablette graphique
— Recopier un écran EGA



9 782
ISBN 2
Code 12



F 160/88/..